
Electronic Thesis and Dissertation Repository

7-5-2017 12:00 AM

SIGNET: A Neural Network Architecture for Predicting Protein-Protein Interactions

Muhammad S. Ahmed
The University of Western Ontario

Supervisor
Dr. Lucian Ilie
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science
© Muhammad S. Ahmed 2017

Follow this and additional works at: <https://ir.lib.uwo.ca/etd>



Part of the [Artificial Intelligence and Robotics Commons](#), and the [Bioinformatics Commons](#)

Recommended Citation

Ahmed, Muhammad S., "SIGNET: A Neural Network Architecture for Predicting Protein-Protein Interactions" (2017). *Electronic Thesis and Dissertation Repository*. 4889.
<https://ir.lib.uwo.ca/etd/4889>

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlsadmin@uwo.ca.

Abstract

The study of protein-protein interactions (PPI) is critically important within the field of Molecular Biology, as proteins facilitate key organismal functions including the maintenance of both cellular structure and function. Current experimental methods for elucidating PPIs are greatly hindered by large operating costs, lengthy wait times, as well as low accuracy. The recent development of computational PPI predicting techniques has worked to address many of these issues. Despite this, many of these methods utilize over-engineered features and naive learning algorithms. With the recent advances in Machine Learning and Artificial Intelligence, we attempt to view this problem through a novel, deep learning perspective. We propose a siamese, convolutional neural-network architecture for predicting protein-protein interactions using protein signatures as feature vectors. In comparison to four leading computational methods, we find that our results are comparable to and, in many cases, surpass the results of these methods. The emphasis of the discussion is to show that there is still much room for improvement in the area of PPI prediction using modern deep learning techniques.

Keywords: Machine Learning, Protein-Protein Interactions, PPI, Protein Signature, Deep Neural Network

Acknowledgements

I would like to thank and acknowledge my supervisor, Dr. Lucian Ilie, whose guidance was instrumental to the completion of this thesis. The perspective, patience, and autonomy which he provided were well balanced and contributed significantly to a positive research experience. The many inspiring talks have also helped shape my views of academia and encourage me to continue thinking like a researcher.

I would also like to thank my lab-mate and friend, Yiwei Li, who provided me with numerous pre-sanitized datasets and complete results from other methods. His help was integral as it saved many weeks of manual testing, as well as my sanity.

Finally, I am grateful to my parents and family who have continually offered words of solace and warmth in times of distress, and continue to be supportive in all of my endeavours.

Contents

Abstract	i
Acknowledgements	ii
List of Figures	v
List of Tables	vii
1 Introduction	1
2 Machine Learning Techniques and Methodologies	4
2.1 Deep Neural Networks	4
2.1.1 Logistic Regression	5
2.1.2 Feed Forward Neural Networks	7
2.1.3 Convolutional Neural Networks	8
2.1.4 Siamese Neural Networks	11
2.2 Optimizing Training and Testing	12
3 PPI Predictions	15
3.1 Biological Background	15
3.2 Previous Prediction Methods	17
3.2.1 Experimental Approaches	17
3.2.2 Computational Approaches	18
3.3 Datasets	24
4 SigNet	26
4.1 Protein Signature	26
4.2 Model Architecture	27
4.3 Implementation	29

4.4	Combined Method	29
5	Additional Methods	30
5.1	ProtVec Based	30
5.2	LSTM Based	32
6	Results and Discussion	35
6.1	Comparison Techniques	35
6.2	Comparing Neural Network Architectures	37
6.3	Results	38
6.3.1	C1 Comparisons	39
6.3.2	C2 Comparisons	47
6.3.3	C3 Comparisons	55
6.4	Discussion	63
7	Conclusion	64
7.1	Future Work	65
	Bibliography	67
	Curriculum Vitae	75

List of Figures

2.1	Machine Learning - an overview	5
2.2	The Sigmoid Function. From deeplearning.net	6
2.3	Logistic Regression	7
2.4	Feed-Forward Neural Network. From coderoncode.com	8
2.5	Edge Detection Convolution Kernel. From timdettmers.com	9
2.6	Convolutional Layer [24]	9
2.7	Max Padding [24]	10
2.8	Convolutional Neural Network [24]	11
2.9	Siamese Neural Network [6]	12
2.10	Overfitting. From mlwiki.org	13
2.11	Rectified Linear Unit. From int8.io	13
3.1	Central Dogma of Biology. From thinglink.com	16
3.2	Flaws in Evaluation [43]	25
4.1	Protein Signature for <i>Q9NZR2</i>	27
4.2	SigNet Architecture	28
5.1	Word Embeddings. From tensorflow.org	31
5.2	ProtVec Based Architecture	32
5.3	Recurrent Neural Networks. From karpathy.github.io	33
5.4	LSTM cell. From colah.github.io	33
5.5	LSTM Based Architecture	34

6.1	ROC Curve. From medcalc.org	36
6.2	Biogrid - C1	39
6.3	HPRD - C1	40
6.4	Innate-Experimental - C1	41
6.5	Innate-Manual - C1	42
6.6	IntAct - C1	43
6.7	MINT - C1	44
6.8	Park and Marcotte - C1	45
6.9	Biogrid - C2	47
6.10	HPRD - C2	48
6.11	Innate-Experimental - C2	49
6.12	Innate-Manual - C2	50
6.13	IntAct - C2	51
6.14	MINT - C2	52
6.15	Park and Marcotte - C2	53
6.16	Biogrid - C3	55
6.17	HPRD - C3	56
6.18	Innate-Experimental - C3	57
6.19	Innate-Manual - C3	58
6.20	IntAct - C3	59
6.21	MINT - C3	60
6.22	Park and Marcotte - C3	61

List of Tables

6.1	AUROC values for Deep Learning Methods	37
6.2	Dataset Sizes	38
6.3	Biogrid - C1	39
6.4	HPRD - C1	40
6.5	Innate-Experimental - C1	41
6.6	Innate-Manual - C1	42
6.7	IntAct - C1	43
6.8	MINT - C1	44
6.9	Park and Marcotte - C1	45
6.10	C1 Averages	46
6.11	Biogrid - C2	47
6.12	HPRD - C2	48
6.13	Innate-Experimental - C2	49
6.14	Innate-Manual - C2	50
6.15	IntAct - C2	51
6.16	MINT - C2	52
6.17	Park and Marcotte - C2	53
6.18	C2 Averages	54
6.19	Biogrid - C3	55
6.20	HPRD - C3	56
6.21	Innate-Experimental - C3	57

6.22	Innate-Manual - C3	58
6.23	IntAct - C3	59
6.24	MINT - C3	60
6.25	Park and Marcotte - C3	61
6.26	C3 Averages	62
6.27	Average over C1, C2, C3	63

Chapter 1

Introduction

Proteins are vital molecules in organisms, as they determine a majority of cellular structure and function [51]. It was widely thought that analyzing only DNA would lead to an understanding of an organism, however a great portion of information in DNA is shared between species. For example, over 90% of active human genes are similar to those of fruit flies and worms [50], so there must be more that is responsible for the vast differences and complexities in organisms. The current understanding is that it is the *interactions* between proteins that account for an organism's complexity. Thus, studying the interactions within the proteome (entire set of proteins expressed by an organism) can give us deeper insights into the structure and function of biological components of such species and what makes them so different. Much work has been done in predicting protein-protein interactions. Broadly, these can be divided into two types of methods - experimental and computational.

The two popular experimental methods for determining protein-protein interactions (PPIs) are Yeast two-hybrid (Y2H) [14] and Tandem Affinity Purification (TAP) [48]. Both of these methods require experiments, which are often time and labour intensive and come with a high associated cost. Furthermore, they have been found to have high false negative and false positive rates [11].

Due to this, experimental methods have been devised in an attempt to increase accuracy while minimizing time and cost.

Computational methods can roughly be grouped into six types of approaches [23] - genome based, evolutionary relationships, protein structure based, domain based, network analysis, and primary sequence based [62]. The biggest issue is that many of these large-network based approaches take too long to compute the *interactome* (sum total of all PPIs in an organism) of species. Primary sequence based approaches, with machine learning, have become more popular in recent times as they generally take less time to run. However, these methods have their own problems, namely that many of them make (often blind) assumptions about how proteins interact and how best to compute feature vectors for proteins.

In this thesis, we compare against four primary sequence based, computational methods, and address some of their shortcomings. Martin et al. [36] were one of the first to use machine learning. They use Support Vector Machines (SVMs) to predict interactions, with protein feature vectors defined as Protein Signatures. PIPE [46] computes a prediction matrix based on similarities between proteins. Ding et al. [10] use an ensemble of machine learning methods and define protein feature vectors using multivariate, mutual information. SPRINT [33] also computes a prediction matrix based on similarities.

This thesis proposes a new deep learning architecture for predicting human PPIs called SigNet. The model is a siamese, convolutional neural-network. We use Martin et al.'s protein signature as protein feature vectors (hence the name SigNet). We test on seven different datasets and find that, depending on the dataset, our method is either comparable to or better than the four aforementioned methods. We also define a consensus method, combining SigNet and SPRINT, and find that its predictive ability is better than all previous methods on all datasets.

The thesis is outlined as follows. Chapter 2 is a primer on the relevant deep learning methods used in our model. Chapter 3 is a primer on the relevant biology and PPI

predictions, as well as a brief description of current experimental and computational methods, and considerations for dataset curation. Chapter 4 is a detailed description of SigNet and the training procedure. Chapter 5 is a discussion and comparison of the methods. Finally, we summarize in Chapter 6 and outline possible future research.

Chapter 2

Machine Learning Techniques and Methodologies

This chapter explains the prerequisite machine learning background relevant to our model. A bottom up approach is used to explain how to construct deep learning architectures and specifically the types of methods used in our model. Additional tools for minimizing loss and optimizing training for time and memory are also described.

2.1 Deep Neural Networks

From a high level, all computational tasks (any task done by a computer) can be thought of as performing a function, or transformation, on some input and producing an output. Traditional algorithmic techniques require us to create handcrafted functions with specific rules and parameters for a given task. Machine learning algorithms change this paradigm in that what is programmed is not the function, but a procedure that *learns* a function which can map given inputs to outputs. This idea is expressed in Figure 2.1.

In this section we build an understanding of what deep neural networks are and how they can be used as learning machines.

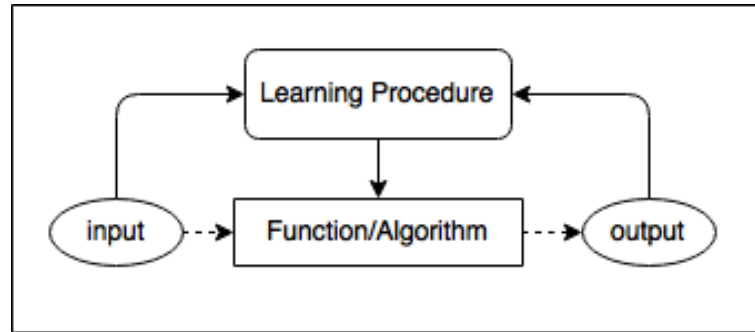


Figure 2.1: Machine Learning - an overview

2.1.1 Logistic Regression

Linear Regression

One of the simplest functions is a linear function. i.e. given some input \mathbf{x} with components $x_1, x_2 \dots x_i$, and corresponding output \mathbf{y} , a simple function consists of assigning a weight θ_i to each x_i . The output of this function is a *hypothesis* and can be written in compact, vector notation as:

$$h_{\theta}(x) = \theta^T x \quad (2.1)$$

Of course, this hypothesis function may not accurately predict the true \mathbf{y} . Thus, we assign a *cost* which tells us how well the hypothesis did. The following is the **least squares** cost for m training samples:

$$J(\theta) = \frac{1}{2} \sum_{j=1}^m (h_{\theta}(x^{(j)}) - y^{(j)})^2 \quad (2.2)$$

Ideally, we wish to choose the value for θ which minimizes the cost. One such procedure for ‘learning’ these parameters is **gradient descent**. We start with an initial guess for θ and use the gradient of the cost, with respect to θ , to update θ such that the cost moves in the direction of steepest descent [41]. For a single training sample, this is defined as:

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad (2.3)$$

Where α represents a learning rate, or the *speed* at which the cost converges to its minimum. This equation is the heart of the learning procedure.

Logistic Regression

Logistic regression is used in the case where the output of the function we wish to approximate is not continuous but a discrete label, $\mathbf{y} \in \{0, 1\}$. It is nearly identical to linear regression, with the only difference being the definition of the hypothesis.

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad (2.4)$$

Here, $g(x)$ is the sigmoid function, also known as the logistic function. It is visualized in Figure 2.2.

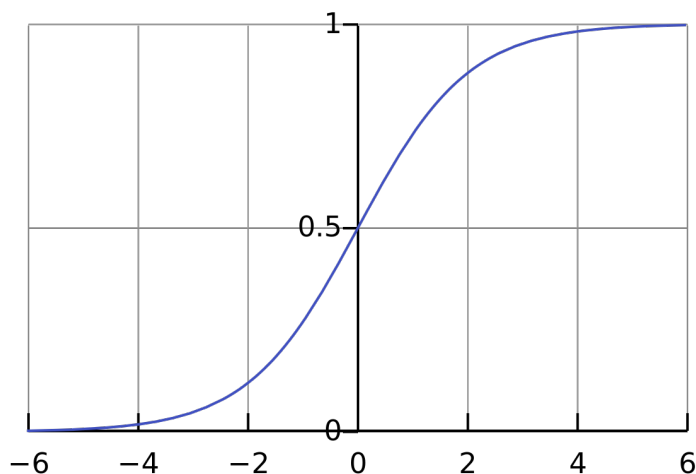


Figure 2.2: The Sigmoid Function. From deeplearning.net

The range of this function is bounded by the range $(0, 1)$, where large values of x will approach 1 and small values will approach 0. We use this continuous function to approximate the discrete label we wish to predict (0 or 1), since a continuous function is

differentiable, whereas a discrete function is not.

A visual overview of logistic regression is shown in Figure 2.3.

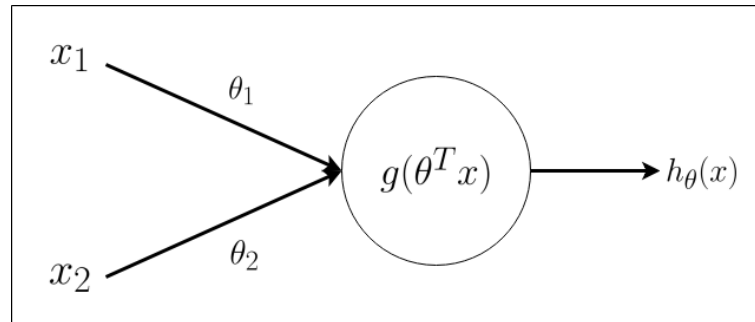


Figure 2.3: Logistic Regression

This method of being given many training samples with their correct labels and learning an approximate function which can map the inputs to the outputs is known as **supervised machine learning** [40].

2.1.2 Feed Forward Neural Networks

One of the earliest formulations of an artificial neural network was proposed by Mulloch and Pitts in 1943 [37]. Called the *perceptron*, this was an attempt at modelling human neurons. Each *neuron* took real values as input and if the sum was greater than some threshold the neuron would ‘activate’ and send a signal to the next neuron. Although this idea of ‘summing over inputs’ sounds similar to previously discussed ideas, the parameters of a perceptron can not be optimized using partial derivatives since the functions are not continuous (they are hard-thresholded values). By adding a logistic unit to each of the neurons, the model becomes amenable to differentiation techniques. Making this modification brings us to the modern Feed-Forward Neural Network. It can be seen as a combination of *logistic regression units* that feed into other such units (Figure 2.4).

Since the final cost is a function of the weights, θ , we can use differentiation, specifically the gradient descent technique, to optimize the weights of the network. The method

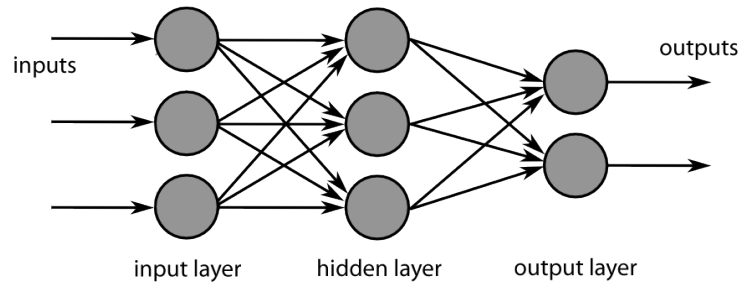


Figure 2.4: Feed-Forward Neural Network. From coderoncode.com

of finding the small changes in θ (*errors*) for each layer that will push the cost towards its minimum is known as **backpropagation** [52].

2.1.3 Convolutional Neural Networks

Introduced by LeCunn et al. in 1999 [30], convolutional neural networks have recently gained significant popularity in image processing tasks ever since they were used to win the ImageNet competition in 2012 by a significant margin [29]. This work is regarded as one of the most influential papers in deep learning, and many advances have been made in convolutional neural networks since then [54, 57].

Intuitively, we can understand convolutional neural networks by the same principle that underlies *convolutional kernels* in computer vision. In this context, a convolution is a real-valued matrix that acts as a sliding window over an image. That is, each pixel in an image is transformed by adding it to its local neighbours, weighted by the values in this *convolution kernel*. We can use image convolutions to perform functions of blurring, sharpening, edge detection, and more [35, 60]. An example of a convolution kernel and its corresponding transformation is shown in Figure 2.5.

Convolutional Layers

A convolutional layer is a type of neural network layer where each neuron is locally connected, instead of fully connected (as in feed-forward networks). For a 2D image input, we define the **width** and **height** of a convolution layer which will act on local regions of

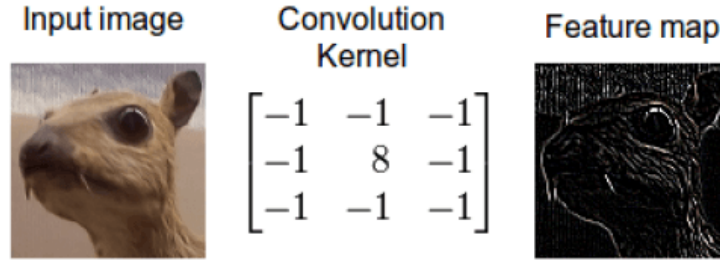


Figure 2.5: Edge Detection Convolution Kernel. From timdettmers.com

the input, and slide over all such regions. It is important to point out that weights are shared as this layer slides across regions. Thus, we can think of this layer as a convolution kernel or *filter*, the weights of which will be learned through backpropagation. We may also choose to learn many such filters at the same time, this gives the convolutional layer **depth**. The output of a convolutional layer will be a new image with given depth that has been transformed through these filters. A visual representation of this is shown in Figure 2.6.

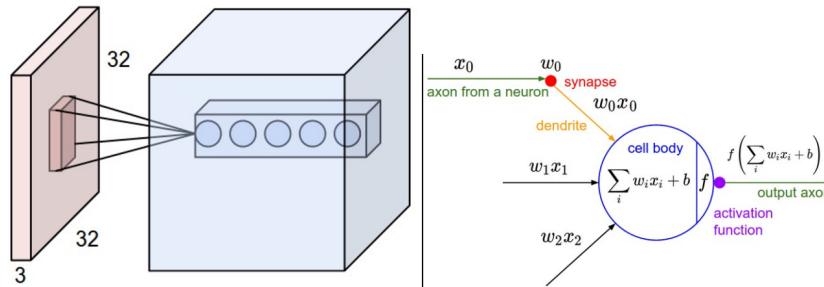


Figure 2.6: Convolutional Layer [24]

In addition to width, height and depth a few other considerations need to be made when implementing convolutional layers.

The **stride** defines how many pixels the filter will ‘jump’ when sliding over the input. For example, a stride of 1 moves 1 pixel at a time, while a stride of 3 will skip 2 pixels after every local region. Larger strides produce smaller output volumes.

Due to how a convolutional layer acts on input, the edges of an image may be cut off.

We can choose to **pad** the edges of the input if we wish to preserve the original width and height.

Finally, an **activation function** is often applied after a convolution layer, similar to how we apply the sigmoid function in each layer of a feed-forward network. A function which has worked well in practice for convolutional architectures is the Rectified Linear Unit (ReLU) [17]. This is explained in more detail towards the end of this chapter.

Pooling Layers

A common type of layer used after convolutional layers is a pooling layer [24]. Similar to convolutions, pooling layers also act on local regions, however their function is to reduce the spatial size of the input in order to decrease the number of parameters that need to be optimized. These layers can be thought of as performing a *downsampling* operation. Common types of pooling include **max pooling**, where the maximum of each local region is used (Figure 2.7), and **average pooling**, where the average is taken.

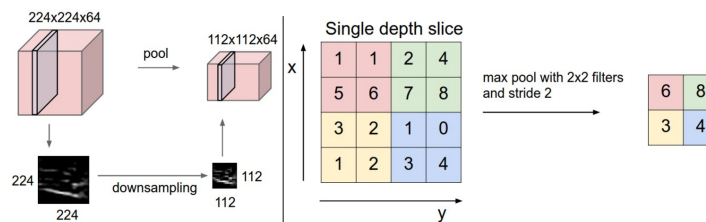


Figure 2.7: Max Pooling [24]

Fully Connected Layers

The final piece to a ConvNet architecture is the fully connected layer. These are often the last layers in the network. These are the traditional feed-forward network layers where every neuron in the input is connected to every neuron in the output. If we view the convolutional and pooling layers as filters on the input, then the fully connected layers can be thought of as ‘summarizing’ the information in the filtered output and producing a final prediction. It is hypothesised that this successive filtering on local regions is similar

to how the human visual system works [19]. An example of a complete ConvNet used for image prediction is shown in Figure 2.8.

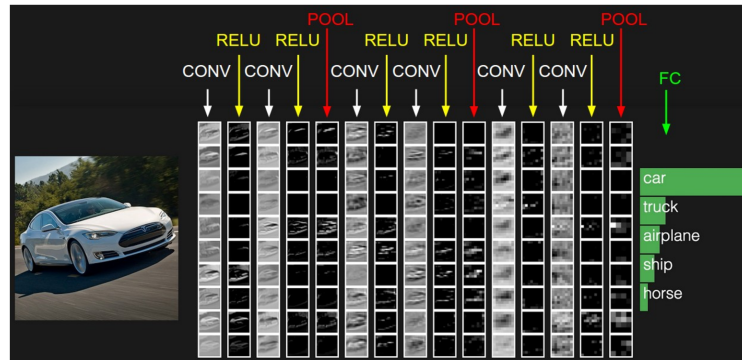


Figure 2.8: Convolutional Neural Network [24]

ConvNets can also be modified to work on 1-dimensional inputs (such as text), as well as 3-dimensional data (fMRI datasets).

2.1.4 Siamese Neural Networks

Siamese Neural Networks [6] are useful when training samples consist of two or more inputs. Siamese networks have shared weights similar to ConvNets, however instead of local regions, weights are shared between entire networks which operate on the respective inputs. More specifically, given two inputs, the same network will operate on both of the inputs, producing two outputs. These outputs are then combined (either by concatenation or another operation) and fed through more layers producing a prediction (Figure 2.9). The weights of this architecture can be optimized using backpropagation.

At a high level, siamese networks can be seen as transforming the inputs in some meaningful way such that a ‘distance metric’, or relationship, between them can be learned. These networks have been successful in various language tasks such as modelling question-answer pairs [61].

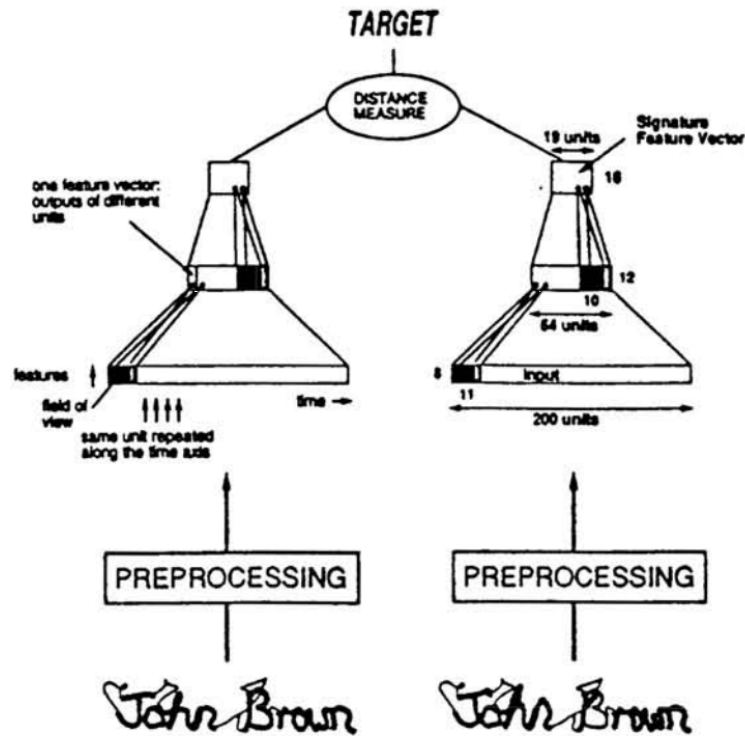


Figure 2.9: Siamese Neural Network [6]

2.2 Optimizing Training and Testing

With the theoretical foundations of neural networks in place, this chapter will outline some practical implementation techniques which need to be considered when optimizing for run time and performance.

Dropout Layers

One issue when training neural network models is that they are very easy to overfit. That is, the network may learn a function that perfectly fits the data, but does not generalize well (Figure 2.10). Adding a dropout layer [56] to the network is one way of tackling this. The dropout layer randomly ‘drops’ (does not use) a proportion of the neurons in a layer during training. This has the effect of forcing the other neurons to learn redundant relationships, and therefore a more general function.

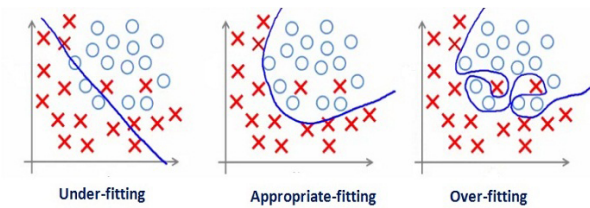


Figure 2.10: Overfitting. From mlwiki.org

Activation Functions

Similar to the sigmoid function, various other activation functions may also be used in neural networks, such as the hyperbolic tangent (\tanh) or the rectified linear unit (ReLU). In practice, ReLUs have been shown to be significantly better for optimizing [15], as well as faster since they are easy to differentiate. A graph of this function is shown in Figure 2.11.

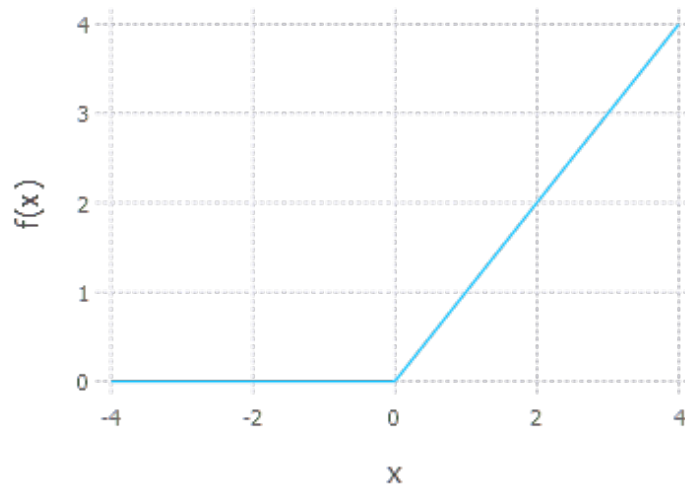


Figure 2.11: Rectified Linear Unit. From int8.io

Optimizers

Backpropagation is a method for finding the gradients of the weights, however it does not actually optimize the weights. Other methods, such as gradient descent, need to be used to actually update the weights according to their gradient. Various techniques have been proposed which work on the principles of momentum [47, 39] and adaptive learning

rates [63] (changing learning rate according to how steep the gradient is). ADAM [27] is a recent method based on both of these principles which has shown be to very effective in practice.

Losses

One of the most important areas to consider when designing a learning-based algorithm is the loss function, since this is the function that is actually being optimized during the learning process. The least-squares cost described above is a simple loss function which does a descent job at comparing the hypothesis to the actual \mathbf{y} values. However, for classification, it gives too much weight to incorrectly labelled samples. A better loss function for this task is the cross-entropy loss. For N training samples, the average cross-entropy loss is defined as:

$$\frac{1}{N} \sum_{i=1}^N [y_i \log h(x_i) + (1 - y_i) \log (1 - h(x_i))] \quad (2.5)$$

Where h is the hypothesis function.

Chapter 3

PPI Predictions

This chapter explains the prerequisite molecular biology background relevant to the task of predicting protein-protein interactions (PPIs). A brief description of previous experimental and computational methods is also given. Finally, four state-of-the-art methods, which we compare our model against, are described in detail.

3.1 Biological Background

It is repeated often that DNA (deoxyribonucleic acid) is the ‘code’ of life - that, in the same way computer code determines the function of the computer, DNA can determine the structure and function of an organism. What is not obvious, however, is why this relationship exists.

DNA is a type of macromolecule known as a *nucleic acid*; it is a molecule found in almost every cell of an organism. Practically, DNA can be thought of as a string of simpler units called *nucleotides*. The four base nucleotides which comprise DNA are Thymine (T), Adenine (A), Guanine (G) and Cytosine (C). When a cell divides, The strands of DNA also replicate themselves. Thus, every cell contains a nearly identical copy. Another function of DNA is that certain parts of it can be *transcribed* into another type of nucleic acid known as RNA (Ribonucleic acid). This is almost identical to the

DNA with the main difference being that Uracil (U) replaces Thymine. The RNA string is then ‘read’ by another type of molecule in a procedure known as *translation*. In this process, every *codon* (set of three consecutive nucleotides) gets translated into a type of molecule called an *amino acid*. The resulting string of amino acids, of which there are twenty types, eventually folds into a 3-dimensional structure called a *protein*.

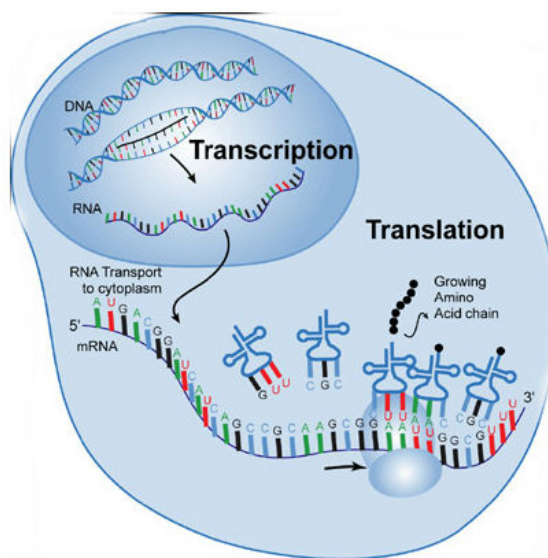


Figure 3.1: Central Dogma of Biology. From thinglink.com

The protein is another type of macromolecule which can be thought of as the ‘functional’ unit of the cell. These molecules are responsible for activities such as cell signaling, metabolic processes, transporting molecules, and maintaining structure. Many of these processes require proteins to interact with other proteins. It is widely accepted that these protein-protein interactions (PPIs) are responsible for an organism’s overall form and function [32]. This procedure, from DNA to RNA to Protein is known as the ‘Central Dogma of Biology’ (Figure 3.1) and is the reason we consider DNA as the biological ‘code’ of the organism - because it contains the ‘information’ that is interpreted as proteins, which are responsible for the actual physical processes of the cell. The portion of the

DNA that is expressed as a protein is called a *gene*. The sum of all such regions in an organism is called a *genome* and the sum of all expressed proteins is called the *proteome* of the organism.

From this perspective, it is clear why studying protein-protein interactions is crucial to understanding the biology of organisms. Figuring out the overall interaction network between proteins in an organism can give us valuable insights about the molecular compositions of proteins, how and why they interact, and as a result, how the organism functions as a whole. Due to the importance of this work, there is a growing demand for fast, cheap and accurate PPI prediction methods. We discuss some of the current methods in the following sections.

3.2 Previous Prediction Methods

Broadly speaking, there are two ways of determining protein-protein interactions, experimentally and computationally.

3.2.1 Experimental Approaches

There are two major experimental approaches for finding PPIs, the yeast two-hybrid (Y2H) approach and tandem affinity purification (TAP).

Yeast Two-Hybrid (Y2H)

Discovered by Fields and Song in 1989 [14], the yeast two-hybrid screening methods detects whether proteins interact by seeing whether they activate a specific, reporter, gene. The premise is that a DNA-binding molecule, called the DNA Binding Domain (DBD), is attached to one protein, and another DNA-binding molecule, called the Activating Domain (AD) is attached to the other protein. When both of these molecules bind to the activating region of a specific gene, that gene is expressed and can be detected.

Since these domains are attached to our respective proteins, the ‘reporter’ gene will only be activated if the proteins interact.

Although it is popular, the Y2H method has the major drawback that it can only detect protein interaction which occur inside the *nucleus* of the cell (a compartment that houses the DNA in cells of more complex organisms). Secondly, this method can only detect one interaction at a time, and has been shown to have high false negative and false positive rates [11, 12].

Tandem affinity Purification (TAP)

Invented by Rigaut et al. in 1999 [49], Tandem Affinity Purification has the advantage that it can detect interacting domains. The main idea is that tags are attached to a protein of interest. These proteins are then extracted from the cell using *beads* which contain molecules that bind to the tags. After the beads are washed (to remove weakly bound molecules), a purification step removes the protein (and its interacting proteins) from the beads. A second purification step removes the tags from the proteins. The interacting complexes can then be identified and analyzed using mass spectrometry.

The biggest drawback of the TAP method is that the tags which are attached to the proteins might actually change their chemistry and how they interact with proteins.

Both of these experimental methods are standard techniques for finding PPIs but suffer from high costs, high false positive and false negative rates, and various other technical limitations [46]. The following section describes computational approaches that have been developed to address some of these issues.

3.2.2 Computational Approaches

The computational methods for predicting PPIs can be grouped into six types of approaches [23].

The **genomic approach** works on the assumption that proteins whose genes are in close proximity to each other are predicted to interact with each other [62].

The **phylogenetic approach** observes the evolution profile of the genes of proteins. It is predicted that proteins that interact will have similar evolutionary histories (in order to preserve their interactions) [62].

Analyzing **protein structures** via x-ray crystallography or NMR spectroscopy can give us insight into specific domains. If proteins contain similar domains, it is possible they function in similar contexts and may interact. A domain is a subsequence (or set of subsequences) in a protein string which has structural or functional significance (i.e. may be responsible for interacting with domains of other proteins).

By the same logic, the **domain based approach** studies domains between proteins, but not necessarily the 3D structure. If two proteins have domains that are similar to two other proteins which interact, we can predict that they will also interact [62].

The **network analysis approach** creates an interaction graph of all the known interactions in an organism. Each protein is a node in the graph, and interactions are represented as edges between the nodes. Small cliques, or sub-graphs, are identified in this graph and missing edges between proteins can be predicted based on their context within the neighbourhood [62].

Finally, the **primary sequence approach** works on the hypothesis that the structure and function of a protein is completely determined by its *primary sequence* of amino acids. Thus, this approach analyzes only the primary sequence of proteins and may employ various string algorithms, or machine learning techniques, to predict which proteins are good candidates for interactions [62].

The hypothesis that underlies the primary sequence approach is the focus of this thesis. If a primary sequence is all that is required to create a protein, then it is assumed that a computational approach should be able to figure out a similar mechanism by relying

on the same information.

To frame this problem more concretely, we will view a protein as a string of characters, where the cardinality of the character set is twenty (20 possible amino acids). From a computational perspective it is a classification problem; the algorithms should take two protein strings as input, and output a binary label indicating whether the two proteins interact or not.

The remainder of this chapter outlines four methods for predicting PPIs using primary sequence. We will compare the results of our model with these methods.

Martin et. al (2005)

The method proposed by Martin et. al [36] was one of the very first methods for PPI prediction using primary sequence. The basic idea is to represent proteins as Signatures [59] and use a Support Vector Machine (SVM) [58] to build a predictive model.

One of the constraints of classical machine learning methods is that inputs have to be represented as meaningful, fixed-length, *feature vectors*. Martin et al. use the protein signature as their fixed-length representation. The protein signature can be defined as $s(A) = \sum_i \sigma_i \mathbf{z}_i$, where A is the protein, \mathbf{z}_i is a basis vector in the signature space \mathbf{R}^N , and σ_i is the count of \mathbf{z}_i . A *signature* consists of an amino acid and its two neighbours, the signature space is all the possible signatures, and the *protein signature* is a vector representing the counts of all the signatures in a protein. For example, $s(LVMTTM) = V(LM) + M(TV) + 2T(MT)$.

It was found experimentally that a signature of three nucleotide bases works better than other sizes. The second contribution of this paper is the usage of a *kernel* function for the SVM. Practically, the kernel can be thought of as a ‘similarity metric’ used by the SVM to better separate positive and negative training samples. They borrow from the string kernel developed by Leslie et al. [31] for use with their signature vector. The one issue, however, is that the input in this case is a pair of vectors, instead of a

single vector. To resolve this issue, Martin et al. define a kernel for a *signature product*, $K((A, B), (C, D)) = (s(A) \otimes s(B)) \cdot (s(C) \otimes s(D))$. They then prove that this is actually equivalent to $k(A, C)k(B, D)$, effectively bringing the computational search space down from R^{N^2} to R^N . Finally, the product is normalized by dividing by the vector magnitudes, $\sqrt{k(A, A)k(B, B)}$.

Martin et al.’s method has been shown to be very effective. Although it is not stated in their paper, their method is very similar to doing a *cosine similarity* metric on a *bag-of-words model* [9], which is a common technique often used in natural language processing tasks.

Ding et. al (2005)

The method of Ding et al [10] also converts each protein pair into a fixed-length feature vector, but uses a random forest classifier [20] for their classification instead of an SVM. The feature vector is defined as the concatenation of two types of feature vectors. The first is created using the multivariate mutual information of each set of 3 amino acids. Secondly, they use the Normalized Moreau-Broto Autocorrelation (NMBAC) of the protein.

Roughly speaking, the multivariate mutual information is a measure of the amount of entropy, or information, contained within a random variable, as it ‘intersects’ with multiple other random variables. To use this method, Ding et al. first categorize each amino acid into 7 groups based on various properties. Then they consider each set of 3 amino acids as a ‘variable’ for which they calculate the MMI. For 3 amino acids, a , b , and c this is defined as:

$$I(a, b, c) = I(a, b) - I(a, b|c) \quad (3.1)$$

In the equation above, $I(a, b)$ and $I(a, b|c)$ are defined as:

$$I(a, b) = f(a, b) \ln \left(\frac{f(a, b)}{f(a)f(b)} \right) \quad (3.2)$$

$$I(a, b|c) = H(a|c) - H(a|b, c) \quad (3.3)$$

Where $f(a)$ is a measure of category a appearing in the protein, and H is a measure of conditional entropies as follows:

$$H(a|c) = -f(a|c) \ln(f(a|c)) \quad (3.4)$$

$$H(a|b, c) = -f(a|b, c) \ln(f(a|b, c)) \quad (3.5)$$

Calculating the MMI for each 3-gram, 2-gram, and individual amino acid categories gives a feature vector of size 119, or 238 when concatenating the vectors of two proteins.

Similar to the MMI method, Ding et al. categorize each amino acid into 6 types based on physiochemical properties before performing autocorrelation. Informally, autocorrelation is a measure of similarity between a signal and a given ‘lag’ of the same signal (or protein in our case) as a function of the lag. Ding et al. perform autocorrelation with 30 different lags for each of the 6 types of categorizations of amino acids, and add the frequency of each of the 20 amino acids, for a total feature vector of size 200, or 400 when concatenating the two proteins.

Finally, the MMI and autocorrelation feature vectors are concatenated for a total of 638 features. This feature vector is then classified using a random forest classifier.

PIPE

Proposed by Pitre et al. in 2006 [46, 44, 45], PIPE is another widely known method for PPI prediction. Unlike previously described algorithms, it does not use machine

learning, rather it exploits the similarity of protein regions between interacting pairs to make predictions.

The first stage in PIPE is the creation of an interaction network - all of the known interactions (from PPI databases) are mapped onto a graph. In the second stage each protein pair, which does not interact, is considered. If regions in the proteins are found to be similar to regions in two other proteins which do interact then those regions of the proteins are said to interact. More concretely, for each non-interacting pair A and B and each interacting pair V and W a sliding window of size twenty is used across all proteins. For each region a_i in A , b_j in B , v_m in V , and w_n in W , if a_i is similar to v_m , and b_j is similar to w_n , a score is increased for those regions of A and B . To keep track of all the scores a matrix can be used, such that location (i, j) in the matrix stores the score of regions a_i and b_j . Finally, if there is a region in the matrix with a high score, above some threshold, then that region is predicted to interact.

PIPE2 [44] and PIPE3 [45] were later introduced with improvements to sensitivity values, as well as a parallel implementation to optimize run time. With the improvements, PIPE will take approximately 3 months to compute the entire human interactome.

SPRINT

SPRINT [33] is a relatively new algorithm for PPI prediction that works on a principle similar to that of PIPE - subsequence similarities between proteins are used to predict which subsequences of a proteins, and thus which proteins, will interact. However, many improvements are made which leads to a better algorithm.

The first improvement is that SPRINT uses spaced seeds to perform matching between subsequences. Roughly speaking, a spaced seed relaxes the constraint that all characters between two strings must match for them to be considered a match, instead it is a mask that defines certain locations which should match. For example, a seed of 1101 means that the first, second and last positions between two 4-character strings should match for

them to be a ‘match’. This technique has been shown to lead to more accurate matches in strings of biological nature (DNA or protein sequences). SPRINT uses an efficient algorithm, SpEED [22], to quickly find multiple seeds which will be used for subsequence matches.

The second improvement is the introduction of a hash table which maps subsequence of proteins to their locations within the protein. This allows them to match similar sections of proteins in constant time, rather than searching all subsequences of all proteins, for every section of a protein. Finally, SPRINT uses the BLOSUM80 substitution matrix [18] to more accurately score matches between amino acids of subsequences.

Similar to PIPE, a PPI prediction is then made if sections of proteins are similar to sections of other proteins which interact.

3.3 Datasets

There are a number of comprehensive datasets available containing known interactions between proteins in organisms. We find that many methods that attempt to use these datasets for PPI prediction have very different ways of processing the data, filtering PPIs, and structuring positive and negative test sets. Furthermore, many of these methods do not even use the same datasets. For this reason, it is often difficult to compare methods with each other. One of the earliest works that attempted a comparative analysis between PPI methods was by Park in 2009 [42]. He found that methods which had good results in their paper, such as those of Shen et al. [53] and Guo et al. [16], did not actually perform that well when compared to other methods, such as those of Martin et al. [36] and PIPE [46] on the same dataset.

Another major issue was found by Park and Marcotte [43] when they noticed that the task of PPI prediction is slightly different from typical classification problems because the

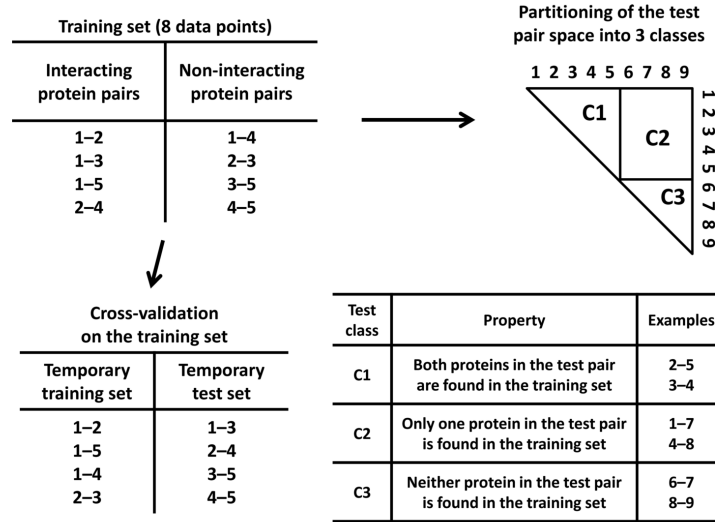


Figure 3.2: Flaws in Evaluation [43]

types of test samples can vary greatly. This is an especially important issue when using machine learning techniques. As shown in Figure 3.2, Park and Marcotte noticed that there can be 3 different types of test samples. When predicting the interaction between two proteins it is possible that both proteins were also in the training set (but in different interactions), they label this type *C1*. It is also possible that one protein was seen in the training set but not the other - this is type *C2*. Finally, type *C3* is when neither protein appeared in the training set.

For these reasons, we consider the four aforementioned methods as their implementations are available and we can compare them using shared datasets. We test all of the methods on 7 different datasets, splitting each of their test sets into the classes *C1*, *C2* and *C3*. This is outlined in more detail in later sections.

Chapter 4

SigNet

In this section, we describe our method of preparing proteins as protein signatures and outline a novel neural network architecture, called SigNet, for PPI prediction. We also describe methods of implementation and training.

4.1 Protein Signature

Majority of neural network models require fixed-length feature vectors as inputs. Since proteins are represented as variable length strings (primary amino acid sequence), we use Martin et al.'s definition of a protein signature to prepare proteins as fixed-length vectors.

As described in Chapter 3, the protein signature is a vector representing the counts of individual amino acid signatures in a protein. A signature is defined as an amino acid and its neighbours. In our case, we use the 2 adjacent neighbours, thus each signature represents a window of 3 amino acids. For example, the individual signatures of the protein *LVMTTM* would be $V(LM)$, $M(TV)$ and $T(MT)$, with counts of 1, 1 and 2 respectively. Since there are 20 possible amino acids, each with 210 different neighbour pairs, the size of our feature vector will be 4,200. For some datasets, there are 23 possible characters (characters representing unknown amino acids), in this case the vector size

will be 6,348.

We impose the additional constraint that the signatures must be ordered in alphabetical order when representing a protein signature vector. This constraint allows us to design a more efficient network by exploiting the fact that similar signatures will have similar counts in a protein. For example, if signature $M(TV)$ shows up many times in a protein, then there is a good chance that $M(TM)$ or $M(TT)$ also appear many times. This was verified empirically by comparing the protein signatures of many proteins.

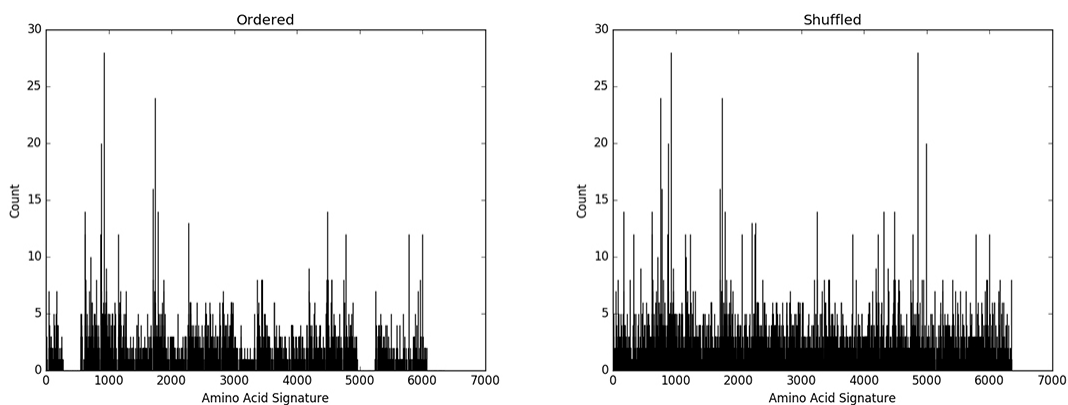


Figure 4.1: Protein Signature for *Q9NZR2*

Figure 4.1 shows a histogram of the 6348 possible signatures in protein *Q9NZR2* when ordered alphabetically, compared with the same histogram when signatures are shuffled. It is clear that alphabetical ordering results in a vector with local similarities. This structure make our feature vector amenable to convolutional techniques, thus allowing us to create a more efficient network with less parameters to optimize.

4.2 Model Architecture

SigNet is a siamese convolutional neural-network. An overview of its architecture is shown in Figure 4.2.

Each protein signature vector first goes through two 1D convolutional layers. Each

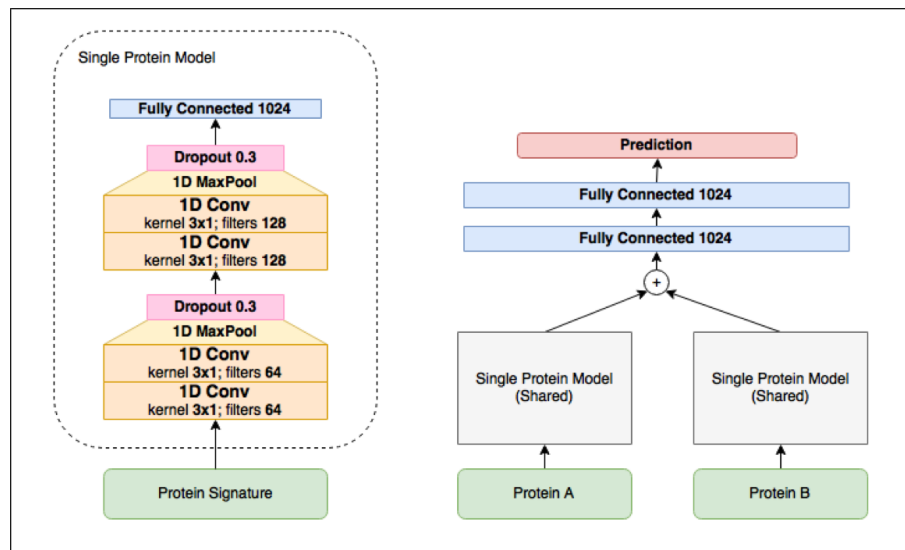


Figure 4.2: SigNet Architecture

of these layers has a kernel size of 3 and a depth (number of filters) of 64. A 1D max pooling operation is then applied with a pool size of 2. A dropout layer is added at this point so that convolutional filters generalize well. A ReLU activation function is used between each of these layers. This pattern of conv-pool-dropout is repeated once more, with the convolutional layers having depth 128. Finally, the output passes through a fully connected layer with 1024 neurons. The purpose of this is to ‘summarize’ the protein signature in some smaller latent vector space.

The entire model described above is applied to the signatures of both proteins in the input, with the weights being shared. The output vectors are then concatenated into a single vector. This passes through two more fully connected layers of size 1024 and ReLU activations.

Finally, the output is compared to the real output value using the cross-entropy loss.

4.3 Implementation

Training and Testing

The ADAM optimizer [27] is used to minimize the loss with an initial learning rate of 0.001. 5 epochs of training are performed for each dataset, with a batch size of 128.

Software and Hardware

SigNet was implemented with Python 2.7.11, using the TensorFlow library [1] for construction of the neural network architecture. Keras [8] was used as a frontend library for simplifying many of the TensorFlow operations. All testing was done on Ubuntu 16.04 LTS.

An NVIDIA GTX 1080 graphics card was used for the computation required during training and testing. Additionally, we use 32gb of RAM and an Intel i7-5930K CPU.

The Python implementation of SigNet, as well instructions on how to install and where to download datasets, can be found at <https://github.com/MrSaad/signet>.

4.4 Combined Method

During testing, we compare SigNet with several other datasets (described in the next chapter). Additionally, we define a *consensus* method in which the scores of SigNet are combined with the scores of SPRINT [33]. We also include the results of this Combined method in the comparisons.

A single-layer, feed-forward neural network with 16 nodes was used to find the optimal weighting scheme between the scores of SigNet and SPRINT for each dataset. The network is trained with cross-entropy loss and the ADAM optimizer.

Chapter 5

Additional Methods

In addition to SigNet, we have tried various other neural-network architectures and feature vector techniques for the PPI prediction task. We describe two of these methods, which were relatively successful, in the following sections.

5.1 ProtVec Based

Embeddings

Assigning fixed-length, real-valued feature vectors to inputs such as words can be tricky since words are *nominal* values - it does not make much sense to assign numerical values to words since they do not have an ordering. That is, one word is not necessarily ‘greater than’, or ‘less than’, another word. In this case we construct a vector of size n , where n is the total number of words in the input space, and assign all values to 0 except one spot which represents the word. This is a **one-hot encoding** representation. This encoding is quite crude since the vector can be very large and sparse, and it does not capture any relationships between words. A word **embedding** aims to resolve these issues by reducing the dimensionality of the vector and representing the word as a *meaningful* feature vector such that semantic relationships between words are preserved.

The embedding technique began developing in the early 2000s with Bengio et al. [4].

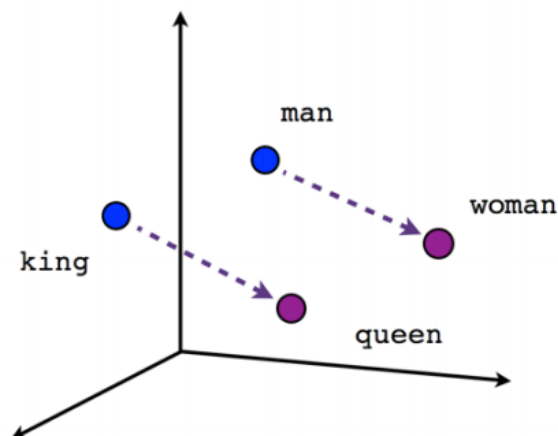


Figure 5.1: Word Embeddings. From tensorflow.org

Since then, many improvements have been made. One of the more recent methods is the skip-gram technique [38] in which word vectors are learned based on the distribution of words in their surrounding neighbourhood. For example, vectors for *man* and *king* will be similar since they appear in a similar gender context, and they will both be a similar distance away from *king* and *queen* respectively (Figure 5.1). Transforming nominal data into a latent vector space in this way is a powerful tool as it captures both contextual as well as semantic information within a feature vector, and therefore generates a significantly better representation of the underlying data.

Model Architecture

Using the idea of word embeddings, Asgari and Mofrad developed an embedding technique for proteins, called ProtVec [2]. Proteins are broken down into ‘words’ of 3 amino acids and, similar to the skip-gram model, an embedding is learned for each word given its surrounding neighbourhood. When visualizing this new vector space they find that amino acids with similar physiochemical properties appear close together, thus this representation is useful. To represent a complete protein as an embedding they take the element-wise sum of all embedded feature vectors in a protein.

We use a siamese neural-network architecture similar to SigNet, but instead of protein signatures as input, we represent proteins as ProtVec embeddings. A visualization of this architecture can be seen in Figure 5.2.

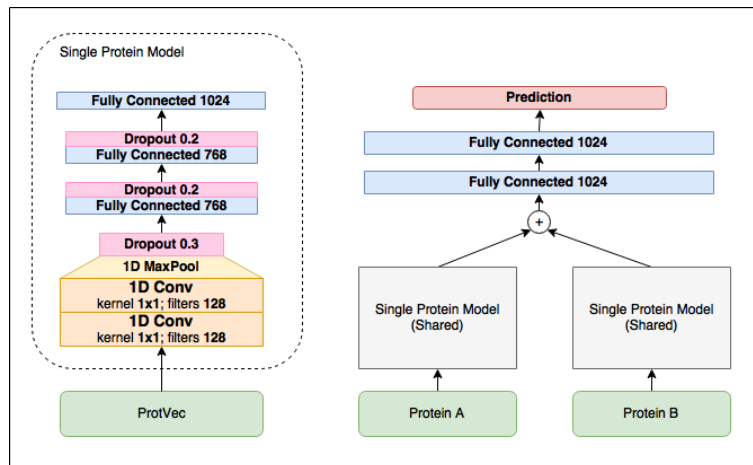


Figure 5.2: ProtVec Based Architecture

The embedding vector is of size 128. A batch size of 64 was used with 30 epochs of training.

5.2 LSTM Based

Recurrent Neural Networks

One of the drawbacks of regular feedforward networks is that they operate on fixed-length feature vectors in which each component has some 'meaning' in the feature space. For example, in the case of SigNet each feature represents the count of a specific amino acid signature. In certain tasks, this representation may not be possible. For example we cannot use raw words as input to feed-forward networks since each words have variable length, and positions of letters don't hold as much meaning as the context within which they appear. Recurrent Neural Networks attempt to solve this problem. In these neural networks each hidden layer is connected to a 'time step' of the input, and also to a hidden layer from the previous time step of the same input. For example, each letter of

a word input would go through the same hidden layer, which keeps track of contextual information since the same layer also receives input from previous letters. Due to this recursive nature, recurrent network layers are quite flexible in how many inputs and outputs they may have (Figure 5.3).

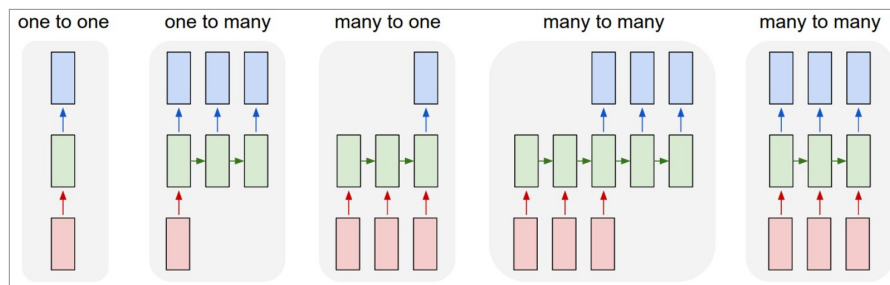


Figure 5.3: Recurrent Neural Networks. From karpathy.github.io

One issue that arises with recurrent networks, however, is the issue of vanishing and exploding gradients. Since a hidden layer keeps track of many previous time steps, its gradient has a tendency to exponentially grow larger or smaller over time, and thus lose information. One way to resolve this is to add additional ‘gates’ or control units to the hidden layer which will learn the appropriate balance between new inputs and long term information. A type of RNN cell that uses this technique is the Long-Short Term Memory [21], or LSTM, cell (Figure 5.4).

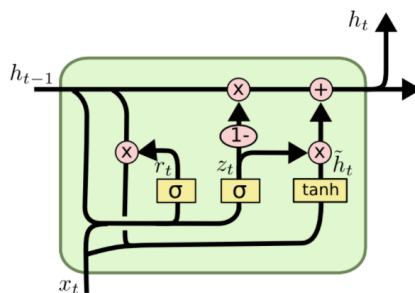


Figure 5.4: LSTM cell. From colah.github.io

Model Architecture

LSTM networks are a natural choice for biological sequences since proteins and DNA can be interpreted as sequences of characters (amino acids and nucleic acids) with contextual importance. Convolutional LSTM networks have already shown to be very effective in various sequence-based prediction tasks, such as predicting subcellular localization of proteins [55].

When using LSTM layers for the task of PPI prediction, we take only the first 200 and last 200 amino acids of each protein as input. Since LSTM networks also suffer from the vanishing gradient problem (to a lesser degree), this prevents us from using entire proteins, which can vary anywhere from 50 to more than 5000 amino acids. Using fixed-length inputs also allows us to take advantage of batch-processing techniques, thus reducing training time significantly. Finally, it has been shown that beginning and ends of proteins contain important sorting information [13], so although we are not using the entire protein, we are still using sections which may be important during interaction.

Each end of the protein goes through a separate stack of conv-pool layers, and finally through an LSTM layer with 512 nodes, the outputs are concatenated. Each protein is sent through this same network, in the same siamese style as SigNet. The full architecture is shown in Figure 5.5.

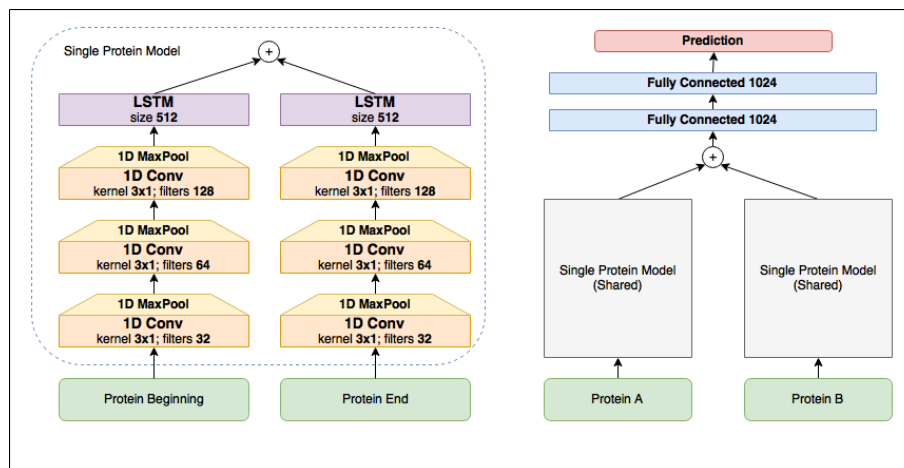


Figure 5.5: LSTM Based Architecture

Chapter 6

Results and Discussion

6.1 Comparison Techniques

In order to compare the results of our method with those of other methods we define the following metrics:

$$TruePositiveRate = Recall = Sensitivity = \frac{TP}{TP + FN} \quad (6.1a)$$

$$FalsePositiveRate = \frac{FP}{FP + TN} \quad (6.1b)$$

$$PositivePredictiveValue = Precision = \frac{TP}{TP + FP} \quad (6.1c)$$

where true positive (TP) is the number of true PPIs predicted as true (correctly), true negative (TN) is the number of false PPIs predicted as false (correctly), false positive (FP) is the number of false PPIs predicted as true (incorrectly), and false negative (FN) is the number of true PPIs predicted as false (incorrectly).

Most binary classification algorithms output real values, given an input. A higher value represents a true prediction, while a lower value represents a false prediction. Thus,

varying the threshold on these output values can give different true positive and false positive rates. We can plot the results of all possible thresholds on a plot known as a Receiver Operation Characteristic (ROC) curve.

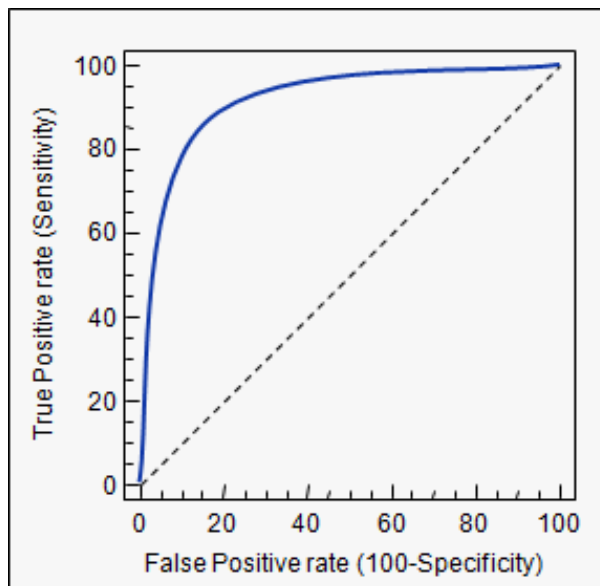


Figure 6.1: ROC Curve. From medcalc.org

Shown in Figure 6.1, a ROC curve shows the change in true positive rate against the false positive rate at varying threshold values. A straight diagonal line represents a classifier with no predictive ability (as good as random guess), while curves that tend closer to the top-left (have larger areas) represent good predictive ability. In particular, it is useful to focus on the beginning of these curves. A good classifier will generally perform better at high specificity values. Thus, steeper curves at the beginning represent better performance.

A similar curve used to model this relationship is the Precision-Recall (PR) curve. This plots Precision against Recall, with varying threshold values. Once again, a larger area represents a better predictive algorithm.

A short ‘summary’ of these curves can be given as Area Under ROC curve (AUROC) and Area Under PR Curve (AUPR) values.

6.2 Comparing Neural Network Architectures

A small comparison was done between our three deep learning methods (SigNet, ProtVec based, and LSTM based) on 4 different databases. We split the test set of each database into the 3 types described by Park and Marcotte - C1, C2, and C3. We compare all of the methods for each test type for each database. This comparison is shown in Table 6.1.

	C1			C2			C3		
	SigNet	ProtVec	LSTM	SigNet	ProtVec	LSTM	SigNet	ProtVec	LSTM
Biogrid	0.94	0.93	0.94	0.86	0.86	0.84	0.77	0.78	0.71
HPRD	0.90	0.86	0.89	0.84	0.81	0.82	0.74	0.75	0.70
Innate-Manual	0.96	0.94	0.89	0.86	0.82	0.79	0.63	0.63	0.64
Innate-Experimental	0.95	0.94	0.95	0.87	0.85	0.85	0.78	0.77	0.72
Average	0.9375	0.9175	0.9175	0.8575	0.835	0.825	0.73	0.7325	0.6925

Table 6.1: AUROC values for Deep Learning Methods

Although all of the deep learning methods perform well, it is clear that SigNet is the better method overall. It outperforms all the other methods on test types C1 and C2, and shows comparable performance on type C3. For this reason, we use SigNet as our primary algorithm when comparing against other, classical algorithms. These comparisons are shown in the following section.

6.3 Results

We compare the results of SigNet, as well as the Combined method (combining SigNet with SPRINT), with the work of Martin et al. [36], PIPE [44], Ding et al. [10], and SPRINT [33].

We train all methods on PPIs from 7 different datasets - BioGrid [7], HPRD [26], two versions of InnateDB [5] (experimentally and manually curated), IntAct [25], MINT [34], and the dataset created by Park and Marcotte upon discovery of the 3 test types (C1, C2, C3) [42].

Table 6.2 shows the number of total PPIs in each of these databases.

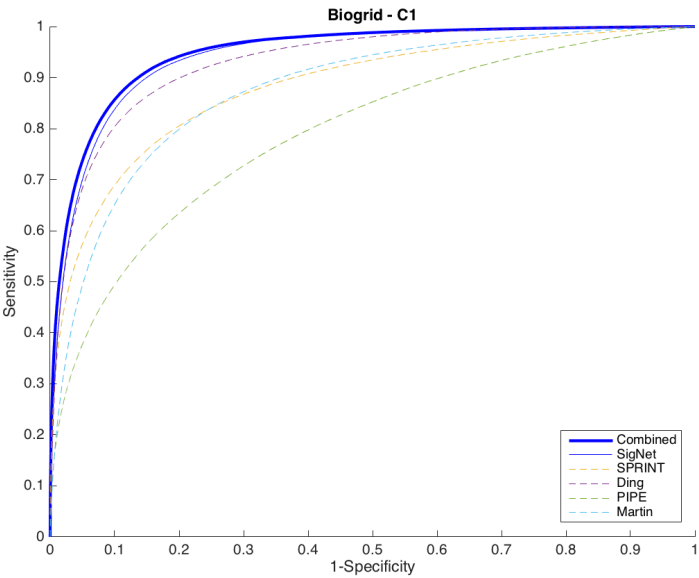
Dataset	Total PPIs
BioGrid	215,029
HPRD	34,044
InnateDB (experimental)	165,655
InnateDB (manual)	9,913
IntAct	111,744
MINT	16,914
Park and Marcotte	24,718

Table 6.2: Dataset Sizes

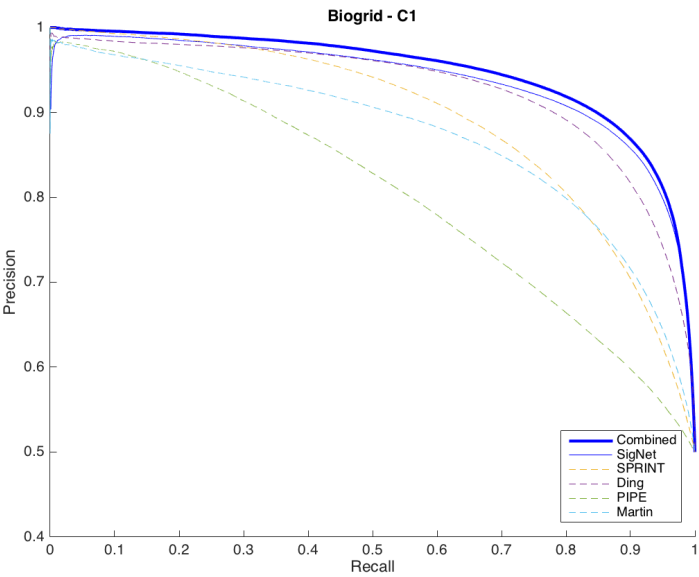
Each dataset was split into 40 splits with 10% being used as the test set. Each test set is further split into the C1, C2 and C3 types. The following sections show ROC and PR curves for each database of each test class. Every plot shows the average of all 40 splits for all methods on one of the seven datasets. The corresponding AUROC and AUPR values are also given along with each plot.

6.3.1 C1 Comparisons

Biogrid



(a) ROC Curves



(b) PR Curves

Figure 6.2: Biogrid - C1

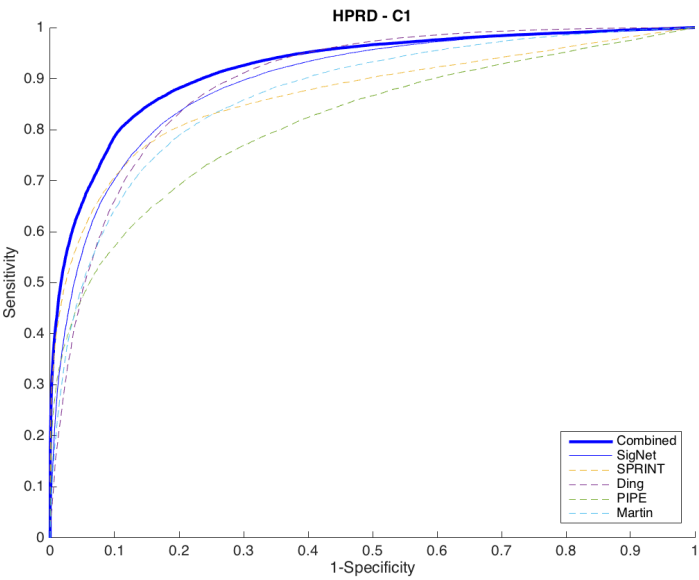
Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
87.54	79.01	93.06	88.11	94.45	94.93	87.20	80.52	93.08	89.24	93.81	94.74

(a) AUROC values

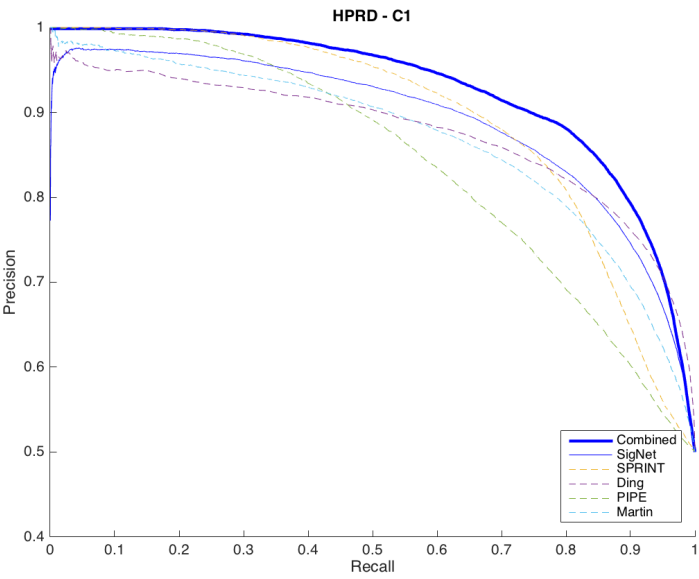
(b) AUPR values

Table 6.3: Biogrid - C1

HPRD



(a) ROC Curves



(b) PR Curves

Figure 6.3: HPRD - C1

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
86.83	81.53	89.34	86.76	89.80	92.05

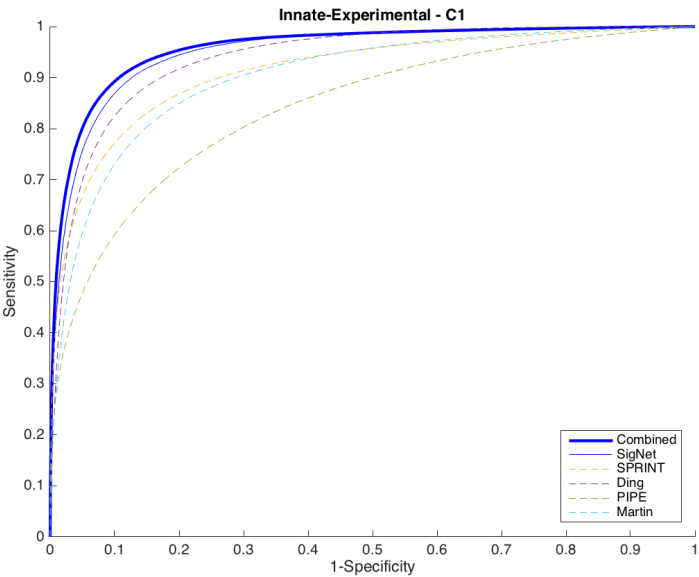
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
86.93	84.31	90.20	89.32	89.51	92.71

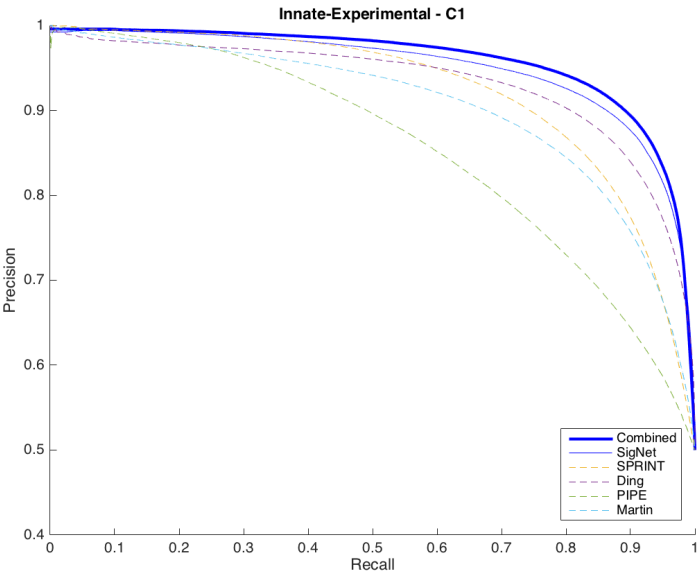
(b) AUPR values

Table 6.4: HPRD - C1

Innate - Experimental



(a) ROC Curves



(b) PR Curves

Figure 6.4: Innate-Experimental - C1

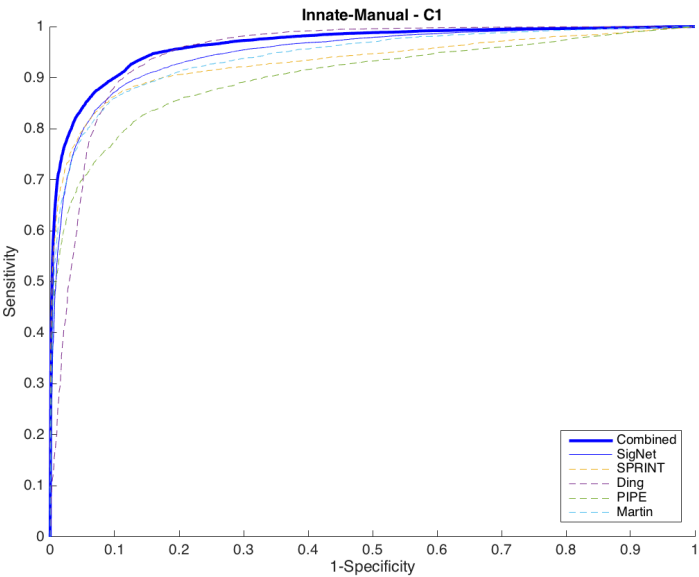
Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
90.18	83.98	93.83	91.34	95.27	95.82	90.31	85.48	94.14	92.25	92.44	95.77

(a) AUROC values

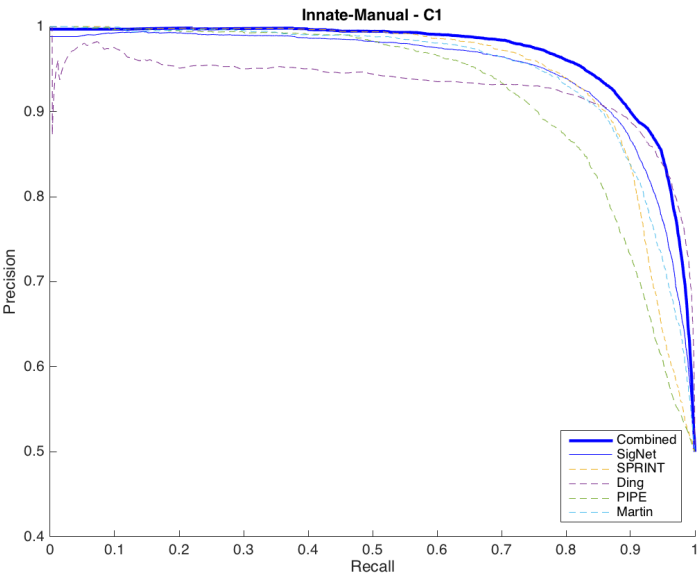
(b) AUPR values

Table 6.5: Innate-Experimental - C1

Innate - Manual



(a) ROC Curves



(b) PR Curves

Figure 6.5: Innate-Manual - C1

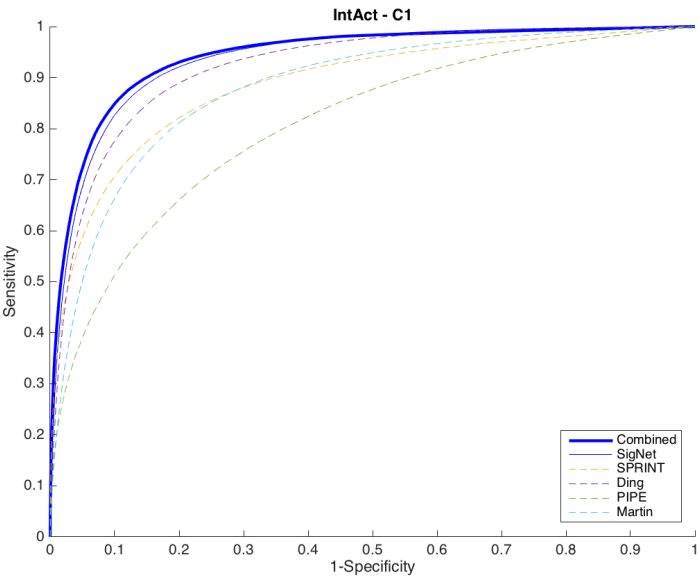
Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
94.11	90.26	94.89	93.09	95.	96.49	94.93	92.22	95.73	94.75	95.07	96.80

(a) AUROC values

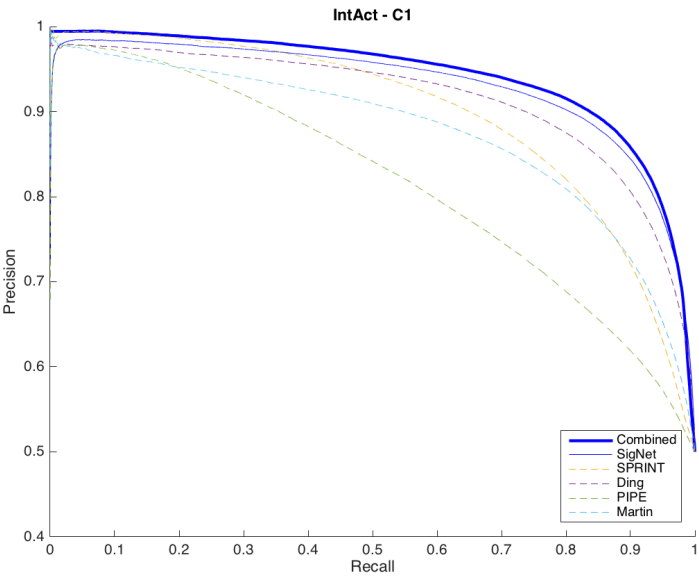
(b) AUPR values

Table 6.6: Innate-Manual - C1

IntAct



(a) ROC Curves



(b) PR Curves

Figure 6.6: IntAct - C1

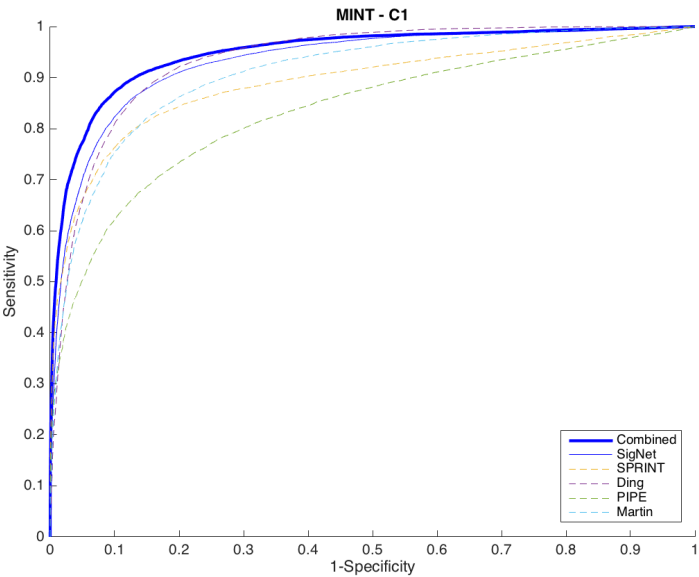
Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
88.02	80.72	92.18	88.69	93.51	94.23	87.51	81.68	92.31	89.71	93.18	94.11

(a) AUROC values

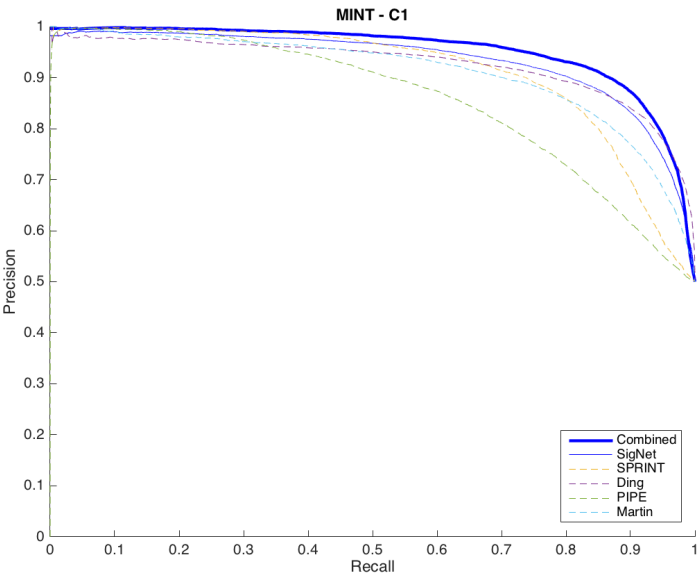
(b) AUPR values

Table 6.7: IntAct - C1

MINT



(a) ROC Curves



(b) PR Curves

Figure 6.7: MINT - C1

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
90.86	83.41	93.54	89.03	93.96	94.91

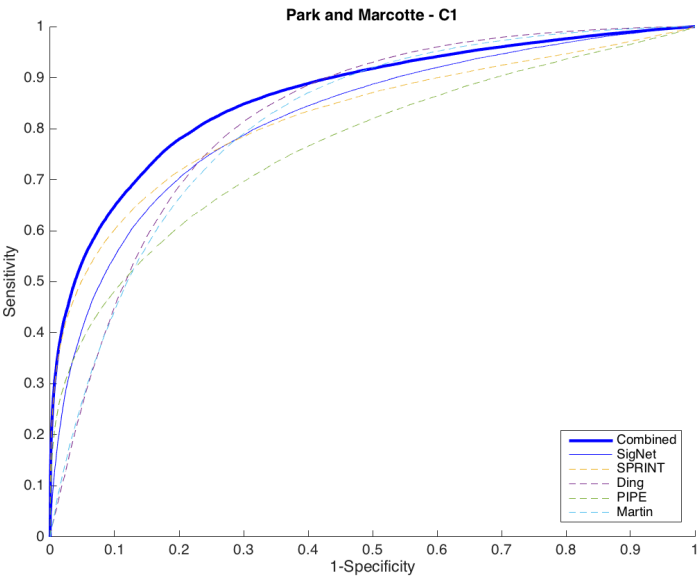
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
91.08	85.93	94.11	91.13	91.76	95.20

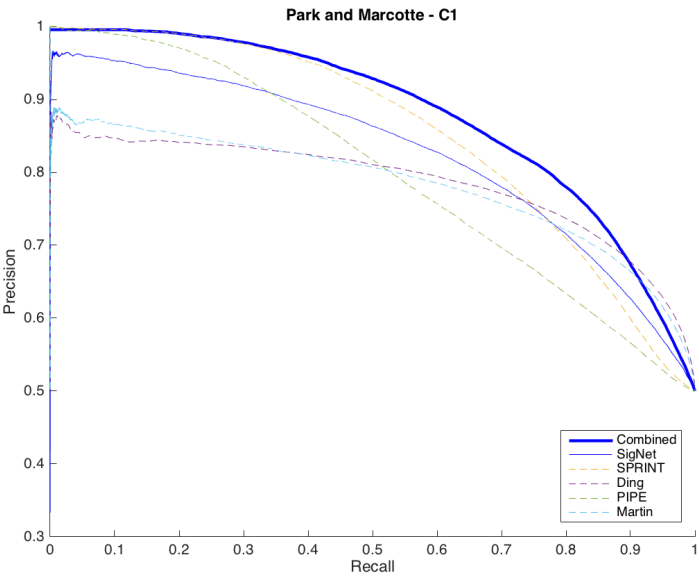
(b) AUPR values

Table 6.8: MINT - C1

Park and Marcotte



(a) ROC Curves



(b) PR Curves

Figure 6.8: Park and Marcotte - C1

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
81.49	76.74	82.00	82.35	82.26	86.46

(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
82.32	79.90	83.00	85.39	82.75	88.01

(b) AUPR values

Table 6.9: Park and Marcotte - C1

Average C1 Scores

It is clear from the above comparisons that SigNet is comparable to and, in many cases, outperforms previous methods on C1 test types. In general, we see that training on larger datasets, such as Biogrid (Figure 6.2), leads to better performance. The Combined method outperforms all methods in all of the datasets.

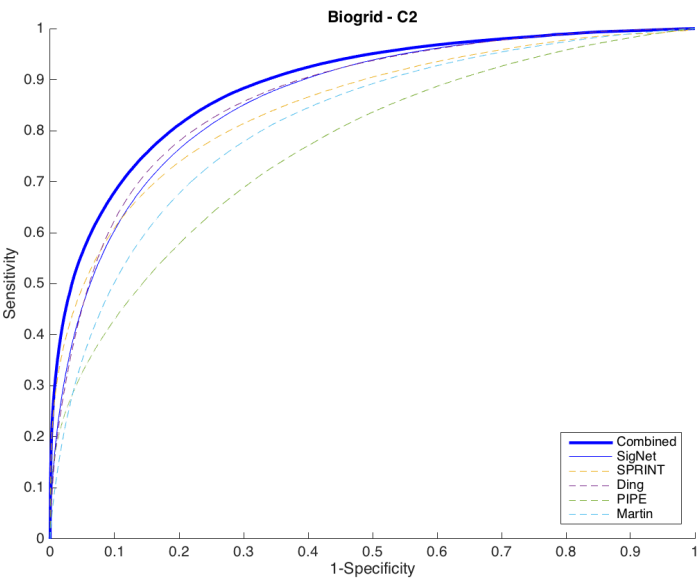
An average of all C1 scores for the different methods is shown in Table 6.10.

Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
88.43	82.24	91.26	88.48	92.15	93.56	88.61	84.29	91.80	90.26	91.22	93.91
(a) AUROC values						(b) AUPR values					

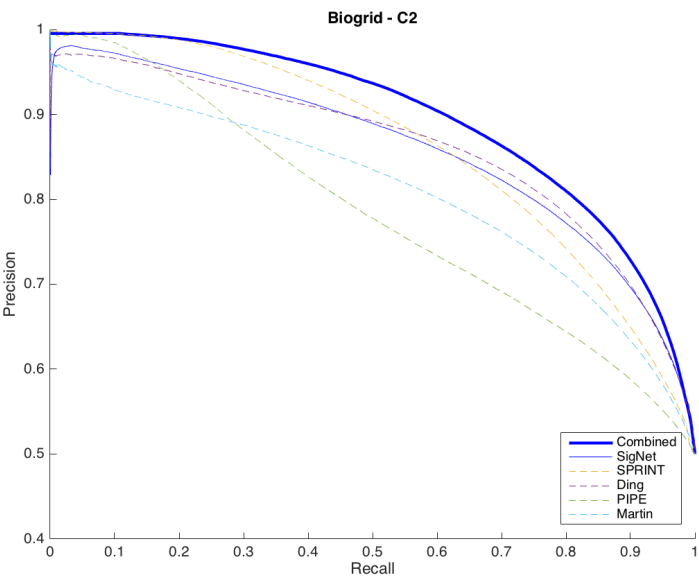
Table 6.10: C1 Averages

6.3.2 C2 Comparisons

Biogrid



(a) ROC Curves



(b) PR Curves

Figure 6.9: Biogrid - C2

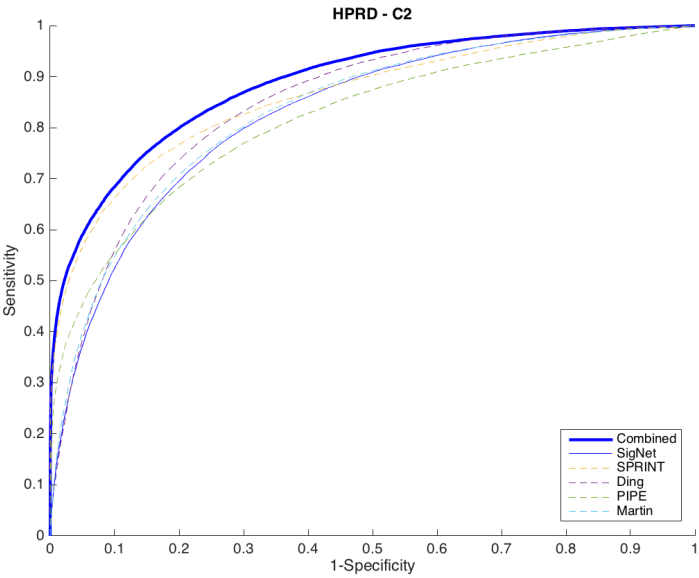
Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
81.33	76.66	86.57	84.67	86.37	88.90	80.76	78.25	86.12	86.30	85.98	89.50

(a) AUROC values

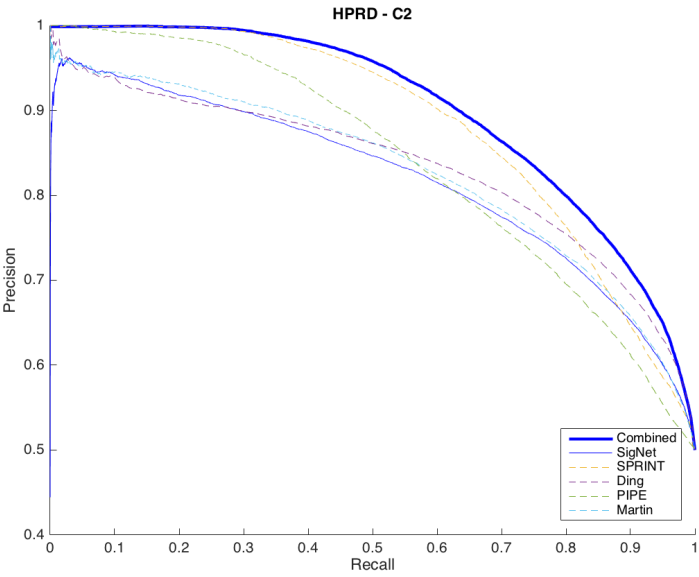
(b) AUPR values

Table 6.11: Biogrid - C2

HPRD



(a) ROC Curves



(b) PR Curves

Figure 6.10: HPRD - C2

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
83.30	81.55	84.78	86.09	82.63	88.93

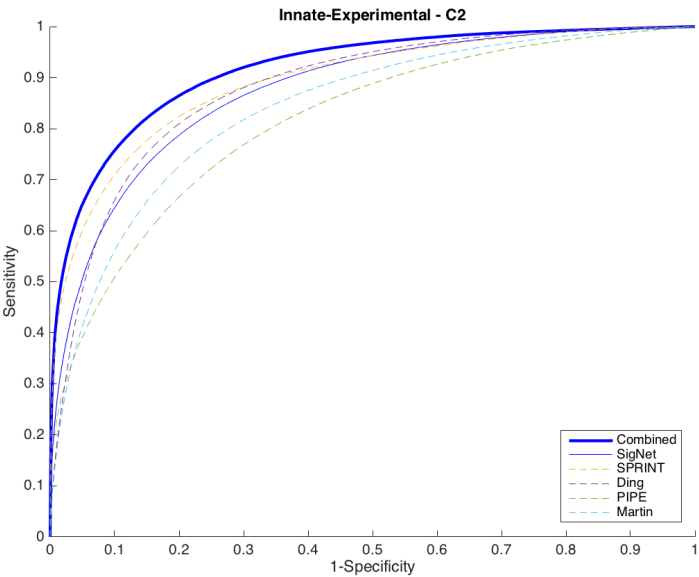
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
82.85	83.98	84.85	88.37	82.30	90.11

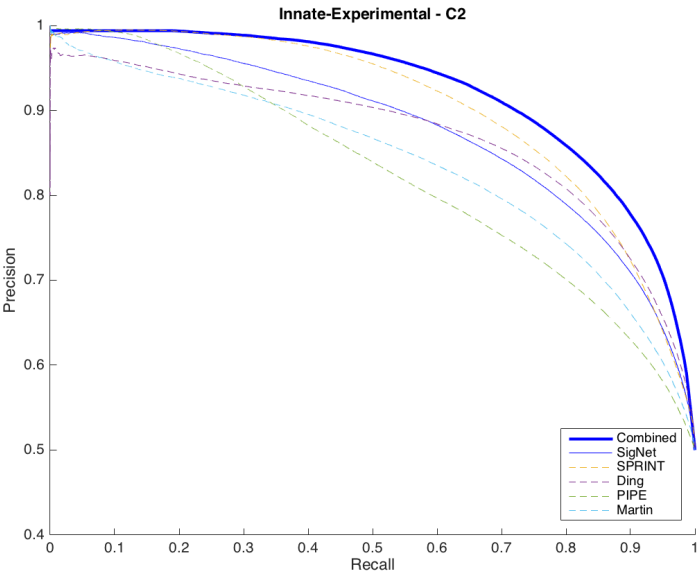
(b) AUPR values

Table 6.12: HPRD - C2

Innate - Experimental



(a) ROC Curves



(b) PR Curves

Figure 6.11: Innate-Experimental - C2

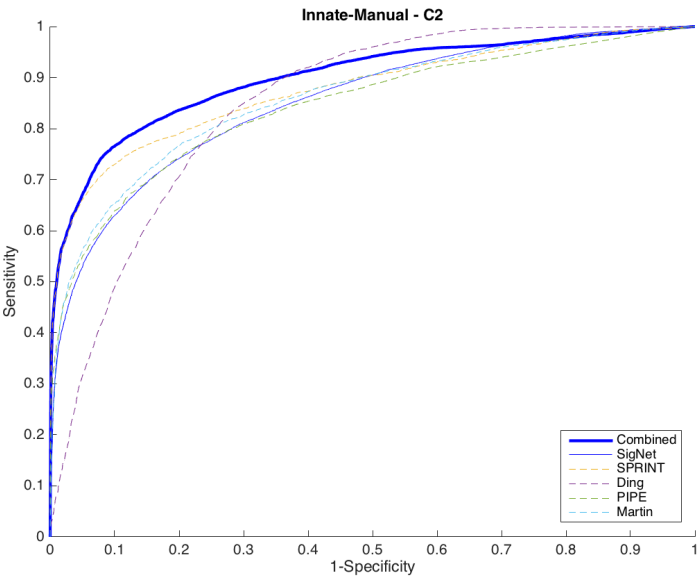
Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
83.96	81.46	87.98	89.31	87.68	91.68	83.74	82.57	87.91	90.37	86.48	92.17

(a) AUROC values

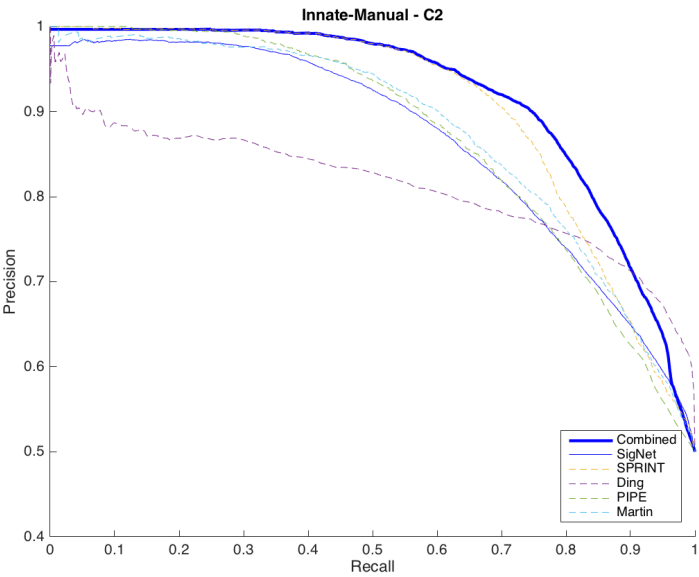
(b) AUPR values

Table 6.13: Innate-Experimental - C2

Innate - Manual



(a) ROC Curves



(b) PR Curves

Figure 6.12: Innate-Manual - C2

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
85.87	84.43	84.74	87.64	87.66	90.06

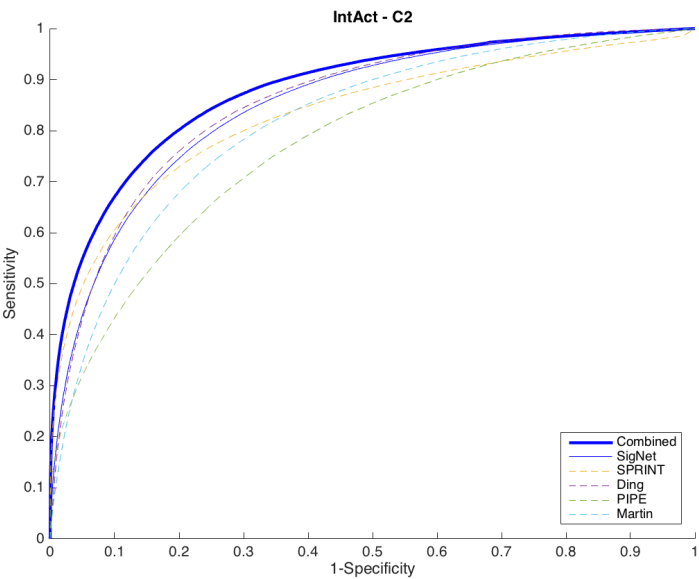
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
87.71	87.22	87.10	90.33	85.16	91.79

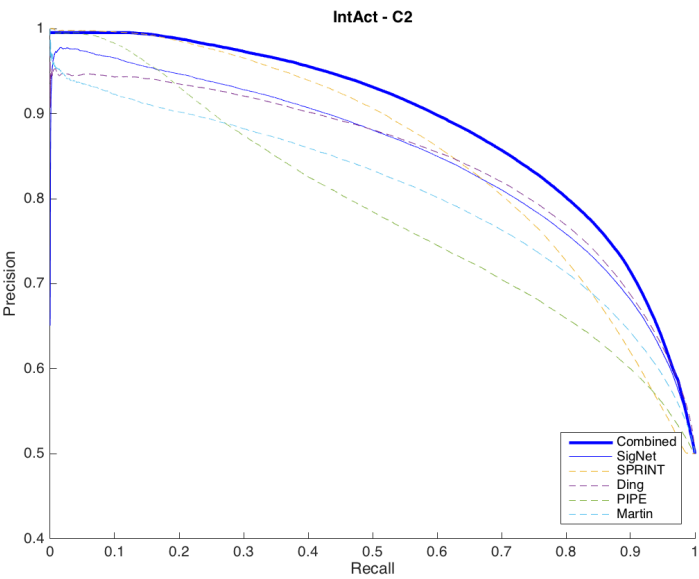
(b) AUPR values

Table 6.14: Innate-Manual - C2

IntAct



(a) ROC Curves



(b) PR Curves

Figure 6.13: IntAct - C2

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
81.687	77.64	85.63	83.14	85.58	88.11

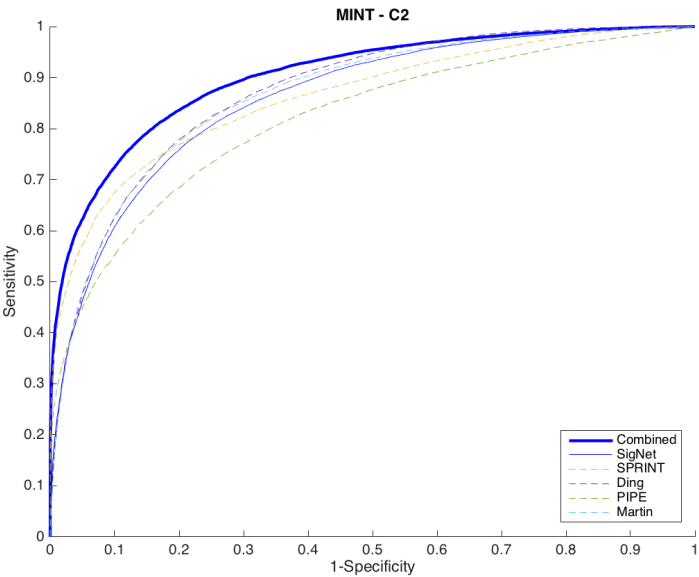
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
80.68	78.69	85.20	85.58	85.17	88.92

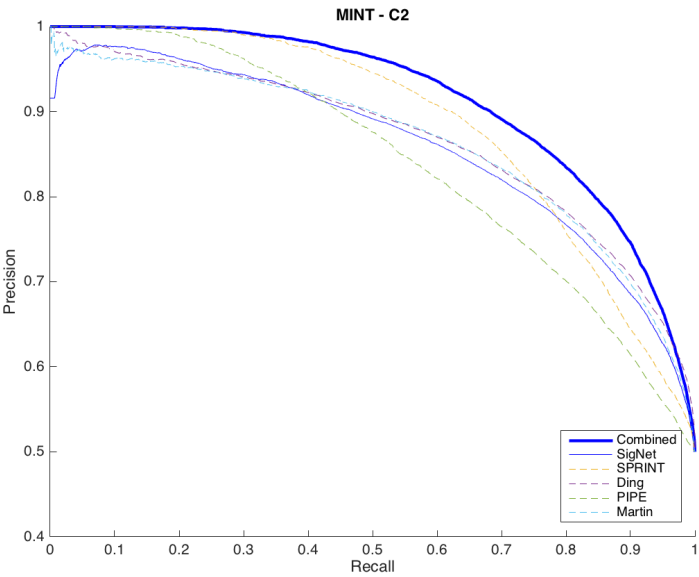
(b) AUPR values

Table 6.15: IntAct - C2

MINT



(a) ROC Curves



(b) PR Curves

Figure 6.14: MINT - C2

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
86.66	81.76	87.17	86.20	84.60	90.37

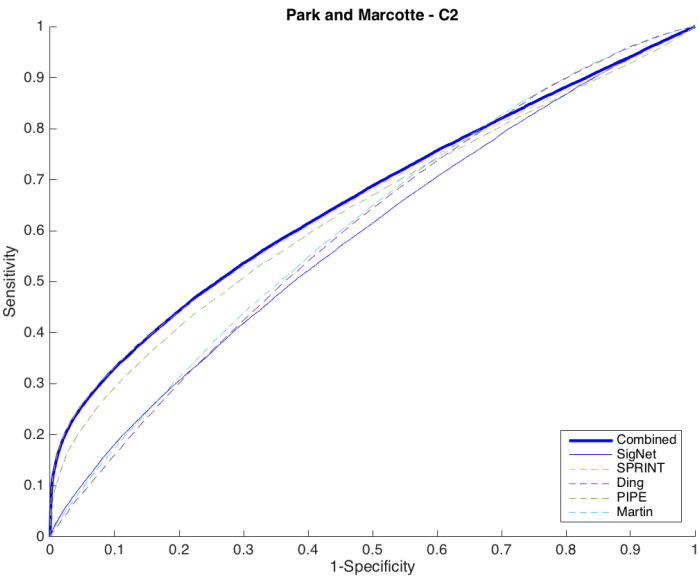
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
86.37	84.08	87.47	88.47	85.46	91.35

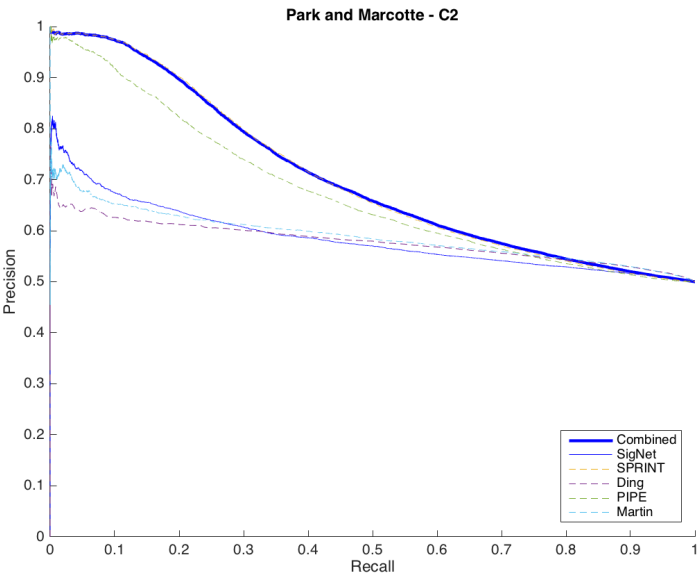
(b) AUPR values

Table 6.16: MINT - C2

Park and Marcotte



(a) ROC Curves



(b) PR Curves

Figure 6.15: Park and Marcotte - C2

Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
60.67	63.76	60.00	65.52	58.69	65.96	60.43	67.41	60.00	70.25	58.60	70.34

(a) AUROC values

(b) AUPR values

Table 6.17: Park and Marcotte - C2

Average C2 Scores

Although SigNet falls behind on the C2 test type for some datasets, the trend is similar to C1 in that larger datasets yield better results. On larger datasets, SigNet is comparable to, and outperforms, previous methods. The Combined method outperforms all methods in all of the datasets.

An average of all C2 scores is shown in Table 6.18.

Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
80.50	78.18	82.41	83.23	81.89	86.29	80.36	80.32	82.67	85.67	81.31	87.74

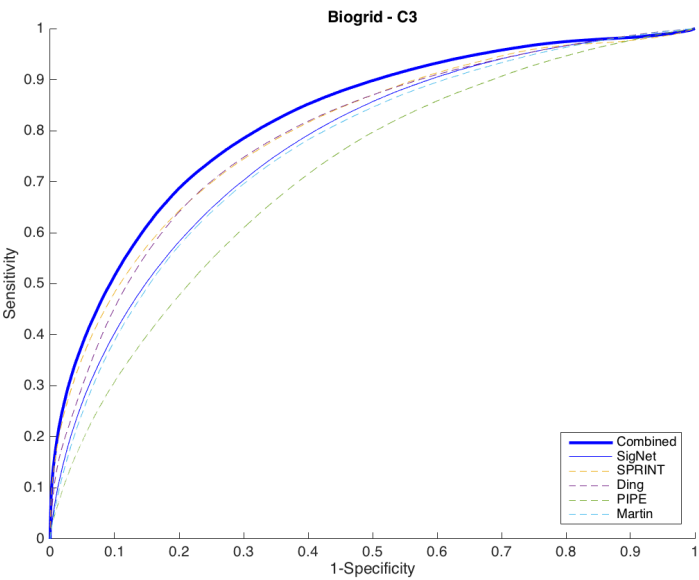
(a) AUROC values

(b) AUPR values

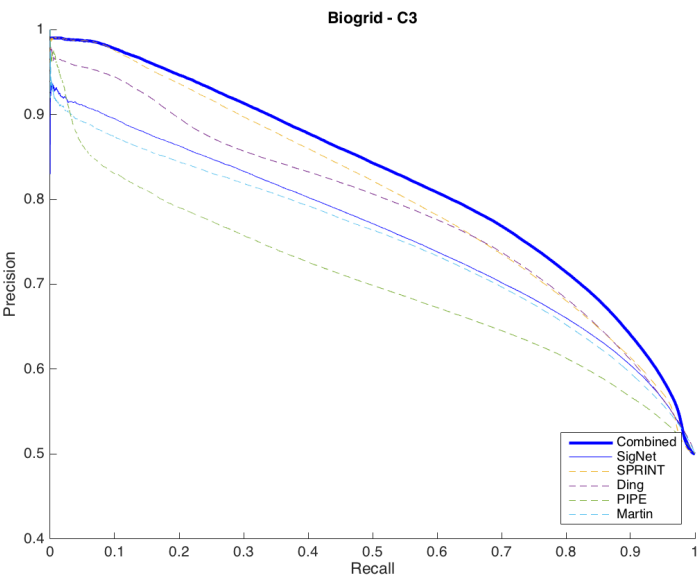
Table 6.18: C2 Averages

6.3.3 C3 Comparisons

Biogrid



(a) ROC Curves



(b) PR Curves

Figure 6.16: Biogrid - C3

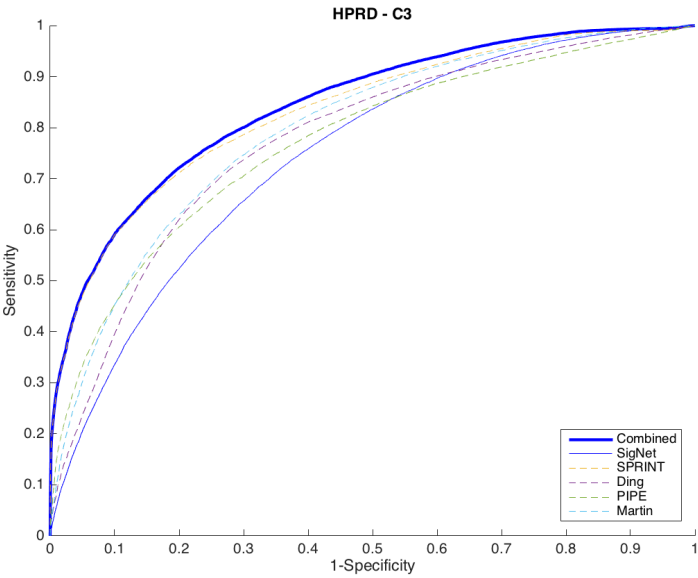
Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
76.20	71.38	79.16	79.67	77.30	82.08	74.89	70.25	77.24	80.59	76.16	82.52

(a) AUROC values

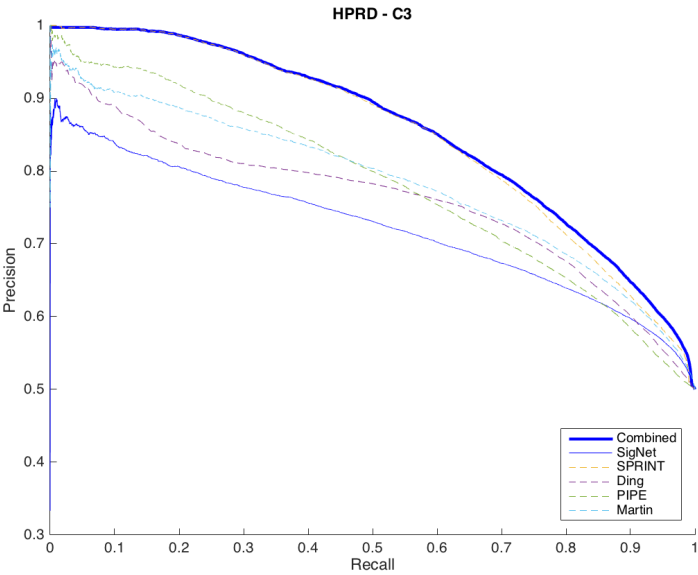
(b) AUPR values

Table 6.19: Biogrid - C3

HPRD



(a) ROC Curves



(b) PR Curves

Figure 6.17: HPRD - C3

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
79.46	77.14	77.51	83.27	75.21	84.27

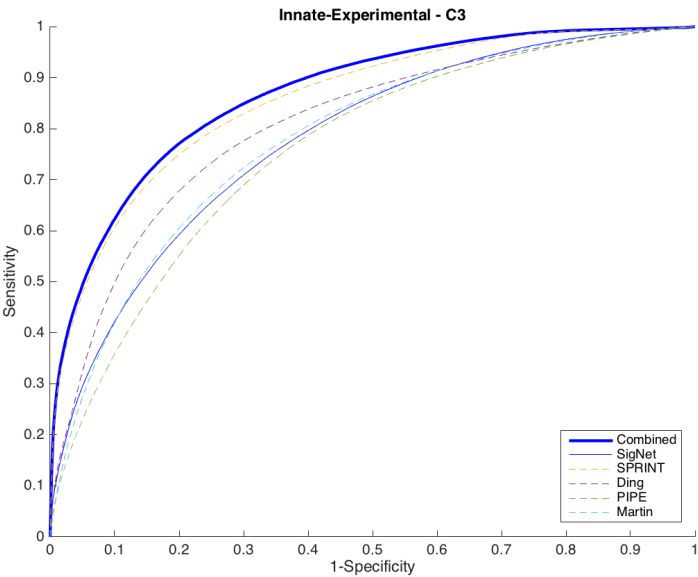
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
78.51	78.28	75.32	85.08	72.55	85.60

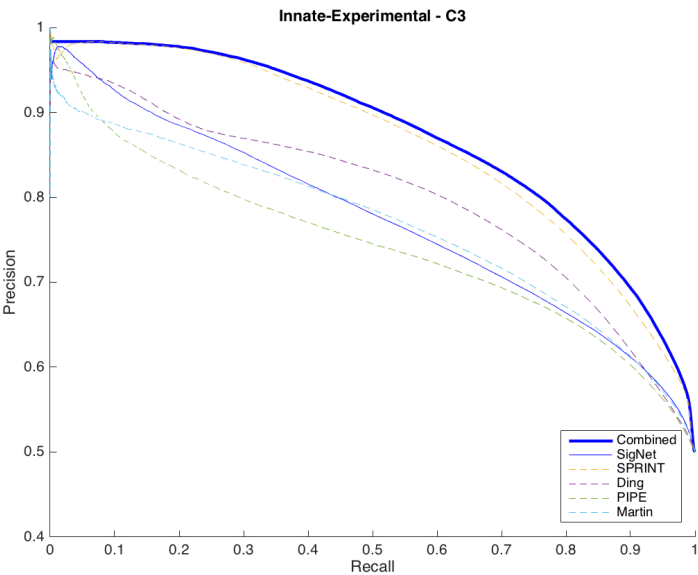
(b) AUPR values

Table 6.20: HPRD - C3

Innate - Experimental



(a) ROC Curves



(b) PR Curves

Figure 6.18: Innate-Experimental - C3

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
78.10	75.89	80.69	85.70	78.29	86.83

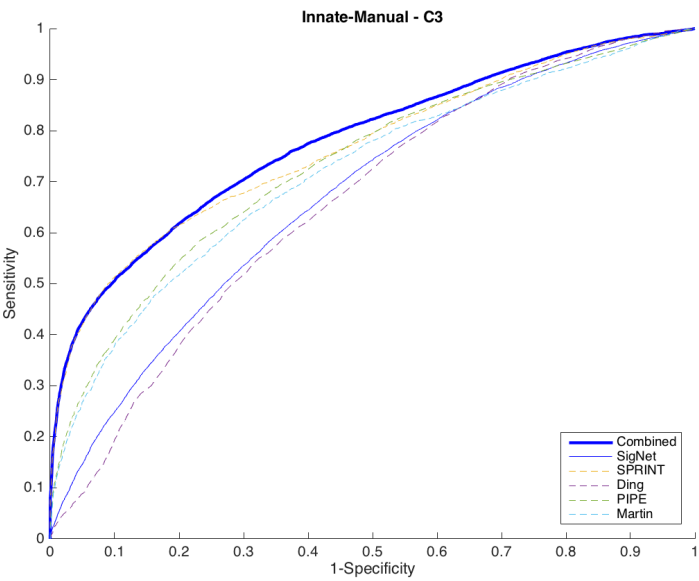
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
76.65	74.42	78.55	86.23	77.33	87.13

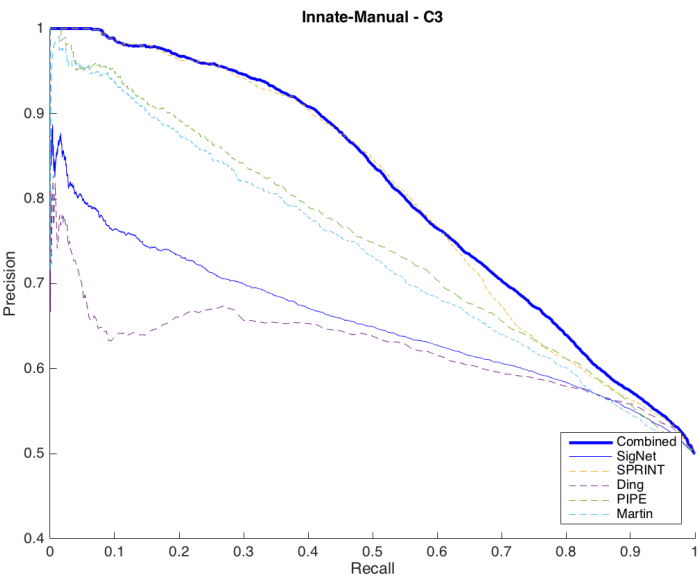
(b) AUPR values

Table 6.21: Innate-Experimental - C3

Innate - Manual



(a) ROC Curves



(b) PR Curves

Figure 6.19: Innate-Manual - C3

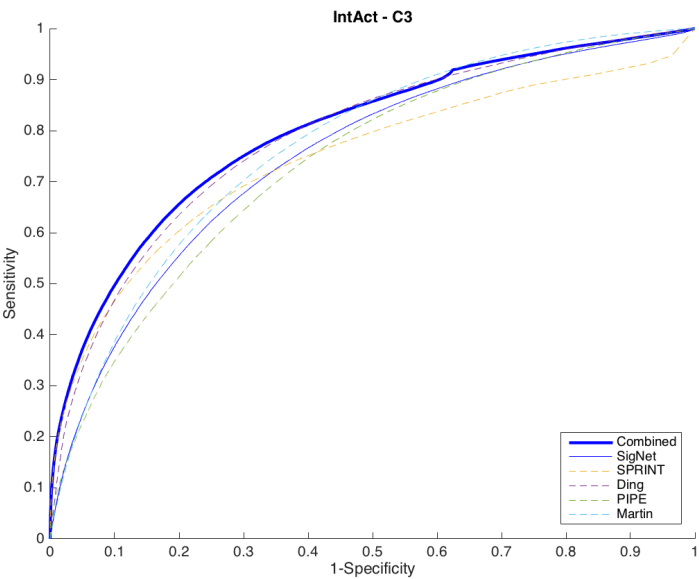
Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
71.75	73.25	65.96	76.57	67.53	77.91	73.49	74.95	66.81	80.17	65.75	80.96

(a) AUROC values

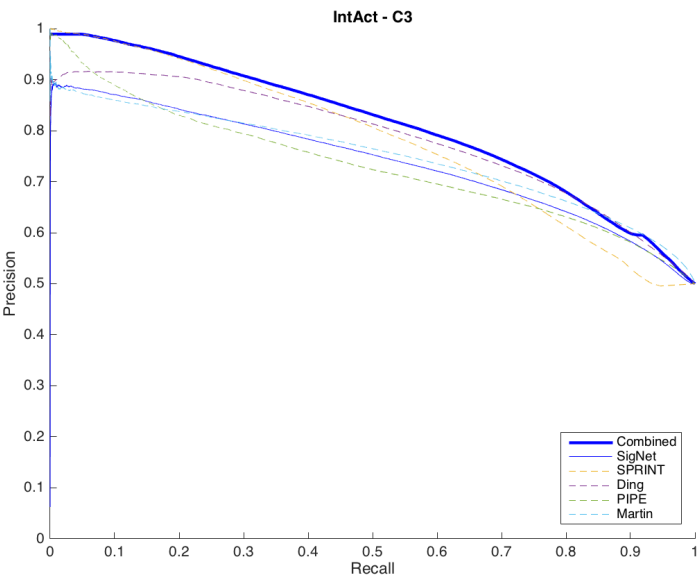
(b) AUPR values

Table 6.22: Innate-Manual - C3

IntAct



(a) ROC Curves



(b) PR Curves

Figure 6.20: IntAct - C3

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
76.94	73.61	78.81	74.44	75.77	79.75

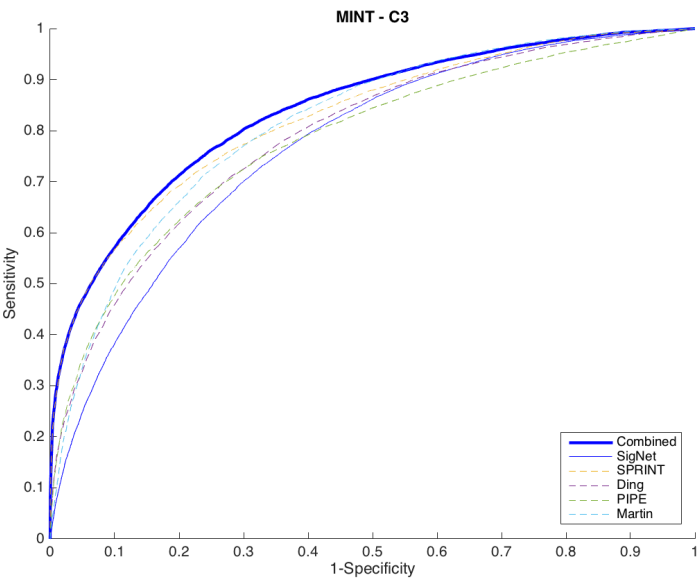
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
74.88	73.11	76.03	78.08	74.10	81.03

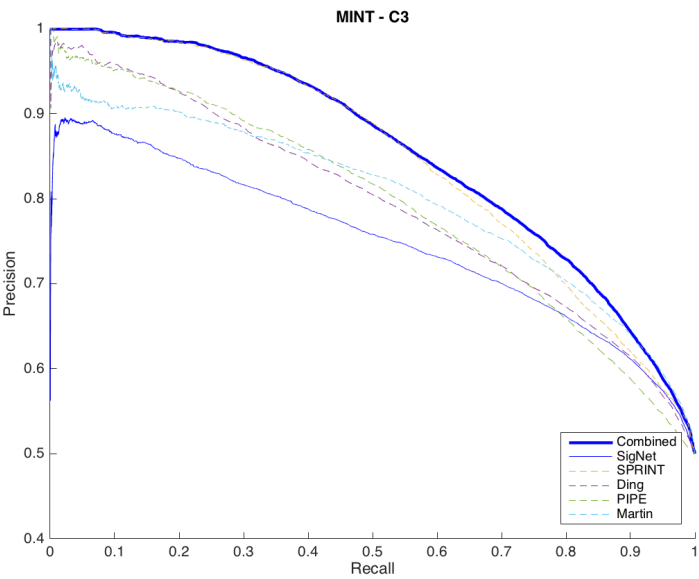
(b) AUPR values

Table 6.23: IntAct - C3

MINT



(a) ROC Curves



(b) PR Curves

Figure 6.21: MINT - C3

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
81.25	78.06	78.94	82.54	76.15	83.91

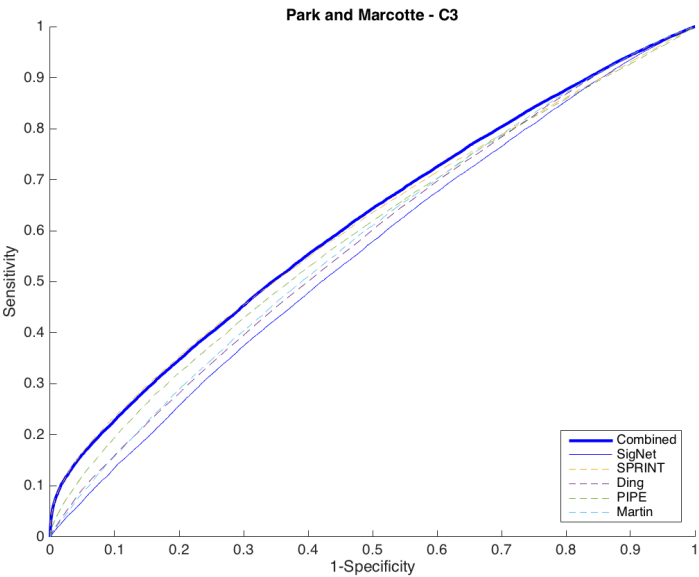
(a) AUROC values

Martin	PIPE2	Ding	SPRINT	SigNet	Combined
80.07	79.28	77.14	84.55	75.10	85.35

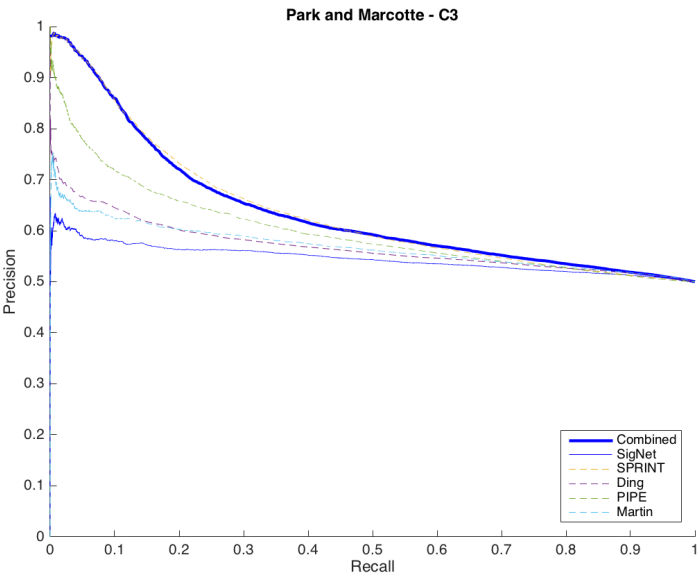
(b) AUPR values

Table 6.24: MINT - C3

Park and Marcotte



(a) ROC Curves



(b) PR Curves

Figure 6.22: Park and Marcotte - C3

Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
57.86	58.90	57.00	60.60	55.67	61.19	57.07	59.84	56.00	63.49	54.56	63.58

(a) AUROC values

(b) AUPR values

Table 6.25: Park and Marcotte - C3

Average C3 Scores

SigNet does not perform as well on C3 test types as it does on C1 and C2. However, following the same trend it performs better than some older methods (PIPE [44] and Martin et al. [36]) for larger datasets. The Combined method performs better than all methods on all datasets.

An average of all C3 scores for the different methods is shown in Table 6.26.

Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
74.51	72.60	74.01	77.54	72.27	79.42	73.65	72.88	72.44	79.74	70.79	80.88

(a) AUROC values

(b) AUPR values

Table 6.26: C3 Averages

6.4 Discussion

Martin	PIPE2	Ding	SPRINT	SigNet	Combined	Martin	PIPE2	Ding	SPRINT	SigNet	Combined
81.15	77.67	82.56	83.08	82.11	86.42	80.87	79.16	82.30	85.22	81.11	87.51
(a) AUROC values						(b) AUPR values					

Table 6.27: Average over C1, C2, C3

As shown in the previous section, on average SigNet outperforms previous methods on the C1 test type. On C2 it outperforms some and is comparable to others, while on C3 SigNet does the worst. When taking the average over all C1, C2 and C3 tests (Table 6.27) we see that SigNet outperforms older methods (PIPE [44] and Martin et al [36]), and is comparable to newer methods (SPRINT [33] and Ding et al. [10]). However, a consistent trend is that SigNet performs better on larger datasets. This result makes sense since deep learning relies on large amounts of data to generalize well.

On average, we also see that a model which combines the scores of SigNet and another state-of-the-art method, SPRINT, outperforms all other techniques on all types of datasets. SigNet performs well on C1 and larger datasets, while SPRINT works well on C3 and across all types of datasets. This suggests that an optimal algorithm could be one that balances classical techniques and machine learning methods, taking the best from both worlds.

Chapter 7

Conclusion

In this thesis we explored the effectiveness of machine learning techniques in predicting protein-protein interactions. We outlined a brief history of machine learning and, specifically, deep learning methods which are becoming increasingly popular. The purpose of this discussion was to show that, with the advent of modern deep learning techniques, there is still much room for improvement in the domain of PPI predictions.

To address the problem using this perspective we developed 3 different siamese neural network architectures. We tested the efficacy of using recurrent layers, modelling proteins as protein embeddings, and modelling proteins as signatures. In the end we found that the architecture based on protein signatures, SigNet, was the best predictive model. We compared against four other leading methods, on seven different datasets, and found that SigNet outperforms them on many test cases, with comparable performance on others. When combining SigNet with a complimentary leading method, SPRINT, we are able to outperform all methods, on all datasets, for all test cases.

This leads us to the conclusion that combinations of deep learning techniques, specifically neural network architectures, can be used to create powerful models which can outperform, or at the very least augment the performance of, classical techniques for PPI predictions.

7.1 Future Work

As deep learning is still a rapidly growing field, there is still much exploratory work to be done in model creation and tuning. We briefly discuss some areas of improvement to our model below.

Recurrent Neural Networks

Since the input to our task is variable-length strings it seems that recurrent neural networks are ideal candidates for this task. This was our initial intuition. Although we experimented with these, it was quickly discovered that they are notoriously difficult to train, and we reported sub-par performance. Given enough computational resources and time, the ideal situation would be to train multiple models with many different hyperparameter settings (size of layers, number of layers, length of input, etc). Due to a limitation in resources, we are unable to train significantly larger models in a practical amount of time.

Another issue with recurrent models is that gradients diminish if there are too many time steps. This can be solved by adding convolutional down-sampling of the input. However, with proteins of lengths varying from 50 to 500 this becomes unfeasible. Adding **attention** [3] to recurrent layers has been shown to solve this problem, as the attention layer learns to pay attention to only select parts of the input. This would allow us to feed the entire protein as input instead of just the beginning and end. However, this was also shown to be difficult to train properly, and without the appropriate resources arriving at the right model would take months.

Alternate Embeddings

We briefly discussed the idea of ‘embedding’ data into some latent vector space. Roughly, the protein signature used by SigNet can be seen as a type of embedding for proteins, where we transform proteins into meaningful vectors. Alternate representations for proteins may be even more useful. Initially, we experimented with various transformations of the protein string, however it may be possible to learn the proper embedding

in an unsupervised manner. **Variational autoencoders** [28] have recently become very popular using this idea. Autoencoders are a type of neural network which have the same input as output. Thus, the hidden layers learn an alternate representation of the data. Since our input varies in length, it may be possible to use the output of a recurrent neural network as input to autoencoders, and thus learn a fixed-length representation of proteins. However, the issue that needs to be overcome is still the same. That is, proteins of significantly large length pose problems for recurrent architectures.

Models and Hyperparameters

We were able to test many neural network architectures of varying size, and experimented with hyperparameter settings using manual and grid searches. However, implementing more robust techniques for finding these parameters, such as bayesian optimization, may allow us to find better models in a reasonable amount of time.

Bibliography

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Ehsaneddin Asgari and Mohammad RK Mofrad. Continuous distributed representation of biological sequences for deep proteomics and genomics. *PloS one*, 10(11):e0141287, 2015.
- [3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

- [5] Karin Breuer, Amir K Foroushani, Matthew R Laird, Carol Chen, Anastasia Sribnaia, Raymond Lo, Geoffrey L Winsor, Robert EW Hancock, Fiona SL Brinkman, and David J Lynn. Innatedb: systems biology of innate immunity and beyond—Recent updates and continuing curation. *Nucleic acids research*, 41(D1):D1228–D1233, 2013.
- [6] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a “siamese” time delay neural network. *IJPRAI*, 7(4):669–688, 1993.
- [7] Andrew Chatr-Aryamontri, Bobby-Joe Breitkreutz, Rose Oughtred, Lorrie Boucher, Sven Heinicke, Daici Chen, Chris Stark, Ashton Breitkreutz, Nadine Kolas, Lara O’donnell, et al. The biogrid interaction database: 2015 update. *Nucleic acids research*, 43(D1):D470–D478, 2015.
- [8] François Chollet. keras. <https://github.com/fchollet/keras>, 2015.
- [9] Cosine Similarity. Cosine similarity — Wikipedia, the free encyclopedia, 2010. [Online; accessed 1-August-2017].
- [10] Yijie Ding, Jijun Tang, and Fei Guo. Predicting protein-protein interactions via multivariate mutual information of protein sequences. *BMC bioinformatics*, 17(1):398, 2016.
- [11] Andrzej Dziembowski and Bertrand Séraphin. Recent developments in the analysis of protein complexes. *FEBS letters*, 556(1-3):1–6, 2004.
- [12] Aled M Edwards, Bart Kus, Ronald Jansen, Dov Greenbaum, Jack Greenblatt, and Mark Gerstein. Bridging structural biology and genomics: assessing protein interaction data with known complexes. *TRENDS in Genetics*, 18(10):529–536, 2002.

- [13] Olof Emanuelsson, Søren Brunak, Gunnar Von Heijne, and Henrik Nielsen. Locating proteins in the cell using targetp, signalp and related tools. *Nature protocols*, 2(4):953–971, 2007.
- [14] Stanley Fields and Ok-kyu Song. A novel genetic system to detect protein protein interactions. *Nature*, 340(6230):245–246, 1989.
- [15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Aistats*, volume 15, page 275, 2011.
- [16] Yanzhi Guo, Lezheng Yu, Zhining Wen, and Menglong Li. Using support vector machine combined with auto covariance to predict protein–protein interactions from protein sequences. *Nucleic acids research*, 36(9):3025–3030, 2008.
- [17] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000.
- [18] Steven Henikoff and Jorja G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.
- [19] Geoff Hinton. Neural networks for machine learning. coursera.org.
- [20] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.
- [21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [22] Lucian Ilie, Silvana Ilie, and Anahita Mansouri Bigvand. Speed: fast computation of sensitive spaced seeds. *Bioinformatics*, 27(17):2433–2434, 2011.

- [23] Matthew Jessulat, Sylvain Pitre, Yuan Gui, Mohsen Hooshyar, Katayoun Omid, Bahram Samanfar, Le Hoa Tan, Md Alamgir, James Green, Frank Dehne, et al. Recent advances in protein–protein interaction prediction: experimental and computational methods. *Expert opinion on drug discovery*, 6(9):921–935, 2011.
- [24] Andrej Karpathy. Convolutional neural networks for visual recognition. Stanford.
- [25] Samuel Kerrien, Bruno Aranda, Lionel Breuza, Alan Bridge, Fiona Broackes-Carter, Carol Chen, Margaret Duesbury, Marine Dumousseau, Marc Feuermann, Ursula Hinz, et al. The intact molecular interaction database in 2012. *Nucleic acids research*, 40(D1):D841–D846, 2012.
- [26] TS Keshava Prasad, Renu Goel, Kumaran Kandasamy, Shivakumar Keerthikumar, Sameer Kumar, Suresh Mathivanan, Deepthi Telikicherla, Rajesh Raju, Beema Shafreen, Abhilash Venugopal, et al. Human protein reference database—2009 update. *Nucleic acids research*, 37(suppl_1):D767–D772, 2009.
- [27] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [28] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [30] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. *Shape, contour and grouping in computer vision*, pages 823–823, 1999.

- [31] Christina Leslie, Eleazar Eskin, Jason Weston, and William Stafford Noble. Mismatch string kernels for svm protein classification. In *NIPS*, volume 15, pages 1441–1448, 2002.
- [32] Emmanuel D Levy and Jose B Pereira-Leal. Evolution and dynamics of protein interactions and networks. *Current opinion in structural biology*, 18(3):349–357, 2008.
- [33] Yiwei Li and Lucian Ilie. Sprint: Ultrafast protein-protein interaction prediction of the entire human interactome. *arXiv preprint arXiv:1705.06848*, 2017.
- [34] Luana Licata, Leonardo Briganti, Daniele Peluso, Livia Perfetto, Marta Iannuccelli, Eugenia Galeota, Francesca Sacco, Anita Palma, Aurelio Pio Nardozza, Elena Santonico, et al. Mint, the molecular interaction database: 2012 update. *Nucleic acids research*, 40(D1):D857–D861, 2012.
- [35] Jamie Ludwig. An introduction to crossbow hunting. Portland State University.
- [36] S. Martin, D. Roe, and J.-L. Faulon. Predicting protein-protein interactions using signature products. *Bioinformatics*, 21(2):218–226, 2005.
- [37] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [38] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [39] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. In *Doklady an SSSR*, volume 269, pages 543–547, 1983.
- [40] Andrew Ng. Neural networks for machine learning. coursera.org.
- [41] Andrew Ng. Cs229 lecture notes. *CS229 Lecture notes*, 1(1):1–3, 2000.

- [42] Yungki Park. Critical assessment of sequence-based protein-protein interaction prediction methods that do not require homologous protein sequences. *BMC bioinformatics*, 10(1):419, 2009.
- [43] Yungki Park and Edward M Marcotte. Flaws in evaluation schemes for pair-input computational predictions. *Nature methods*, 9(12):1134–1136, 2012.
- [44] Sea Pitre, C North, M Alamgir, M Jessulat, A Chan, X Luo, JR Green, M Dumontier, F Dehne, and A Golshani. Global investigation of protein–protein interactions in yeast *saccharomyces cerevisiae* using re-occurring short polypeptide sequences. *Nucleic acids research*, 36(13):4286–4294, 2008.
- [45] Sylvain Pitre. *PIPE: a protein-protein interaction prediction engine based on the re-occurring short polypeptide sequences between known interacting protein pairs*. PhD thesis, Carleton University, 2010.
- [46] Sylvain Pitre, Frank Dehne, Albert Chan, Jim Cheetham, Alex Duong, Andrew Emili, Marinella Gebbia, Jack Greenblatt, Mathew Jessulat, Nevan Krogan, Xuemei Luo, and Ashkan Golshani. PIPE: a protein-protein interaction prediction engine based on the re-occurring short polypeptide sequences between known interacting protein pairs. *BMC bioinformatics*, 7:365, 2006.
- [47] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.
- [48] Guillaume Rigaut, Anna Shevchenko, Berthold Rutz, Matthias Wilm, Matthias Mann, and Bertrand Seraphin. A generic protein purification method for protein complex characterization and proteome exploration. *Nat Biotech*, 17(10):1030–1032, Oct 1999.
- [49] Guillaume Rigaut, Anna Shevchenko, Berthold Rutz, Matthias Wilm, Matthias Mann, and Bertrand Séraphin. A generic protein purification method for pro-

- tein complex characterization and proteome exploration. *Nature biotechnology*, 17(10):1030–1032, 1999.
- [50] Gerald M Rubin and Mark Stoneking. Comparing species From the evolutionary past. 409(February):820–821, 2001.
- [51] Robert F. Schleif. *Genetics and molecular biology*. Johns Hopkins U.P., 1993.
- [52] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [53] Juwen Shen, Jian Zhang, Xiaomin Luo, Weiliang Zhu, Kunqian Yu, Kaixian Chen, Yixue Li, and Hualiang Jiang. Predicting protein–protein interactions based only on sequences information. *Proceedings of the National Academy of Sciences*, 104(11):4337–4341, 2007.
- [54] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [55] Søren Kaae Sønderby, Casper Kaae Sønderby, Henrik Nielsen, and Ole Winther. Convolutional lstm networks for subcellular localization of proteins. In *International Conference on Algorithms for Computational Biology*, pages 68–80. Springer, 2015.
- [56] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [57] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

- [58] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.
- [59] Donald P Visco, Ramdas S Pophale, Mark D Rintoul, and Jean-Loup Faulon. Developing a methodology for an inverse quantitative structure-activity relationship using the signature molecular descriptor. *Journal of Molecular Graphics and Modelling*, 20(6):429–438, 2002.
- [60] Wikipedia. Kernel (image processing) — wikipedia, the free encyclopedia, 2017. [Online; accessed 17-May-2017].
- [61] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *arXiv preprint arXiv:1512.05193*, 2015.
- [62] Javad Zahiri, Joseph Hannon Bozorgmehr, and Ali Masoudi-Nejad. Computational prediction of protein–protein interaction networks: algorithms and resources. *Current genomics*, 14(6):397–414, 2013.
- [63] Matthew D Zeiler. Adadelata: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Curriculum Vitae

Name: Muhammad Saad Ahmed

Post-Secondary Education and Degrees: Western University
London, Ontario
2015 - 2017 M.Sc.

Western University
London, Ontario
2011 - 2015 B.Sc.

Honours and Awards: Western Gold Medal
2015

NSERC - USRA
2014

Related Work Experience: Teaching Assistant
Western University
2015 - 2017

Software Engineer Intern
Google
2016

iOS Developer Intern
RaceRoster
2015